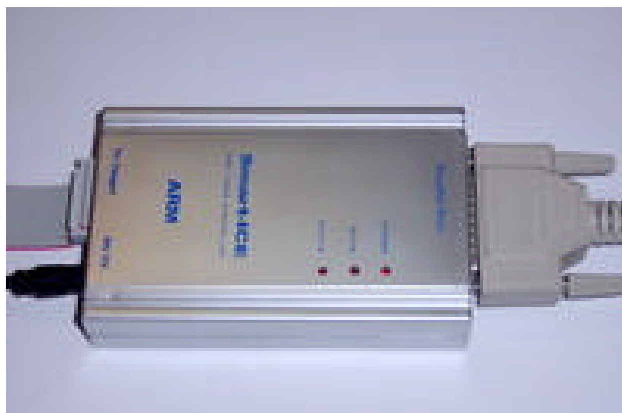
Smart-ICE 使用手册



支持全系列 ARM 内核

支持 RealView Developer Suite

一年保修,终身维修

ARM 专业团队技术支持

上海勤研电子科技有限公司

http://www.armzone.com

Smart-ICE 使用手册

Part 1. 开始使用

本章介绍了如何连接 Smart-ICE 的各个器件，以及如何配置 Multi-ICE server 的软件。包含以下内容：

* 系统要求；
* 连接 Smart-ICE 的硬件设备；
* 连接非标准硬件；
* 启动软件。

1.1 系统要求

本节介绍了 Smart-ICE 的硬件和软件方面的要求：

主机软件要求；

主机硬件要求；

目标硬件要求。

1.1.1 主机软件要求：

在 Smart-ICE 中有两个必需的软件组件：

Multi-ICE server，必须在与仿真器相连的计算机上运行；

Multi-ICE DLL，可以在另一台计算机上运行。

表 1 确定了适用于这些组件的操作系统：

Operating system	Smart-ICE server	Smart-ICE DLL
Windows 95	Yes	Yes
Windows 98	Yes	Yes
Windows Me	Yes	Yes
Windows NT 4.0 (Intel)	Yes	Yes
Windows 2000	Yes	Yes
Solaris 2.6, 7.0, or 8.0	No	Yes
HP-UX 10 or 11	No	Yes
Redhat Linux 6.2 or 7.1	No	Yes

表 1　Smart-ICE 支持的操作系统

由 ADS v1.0.1(或稍后版本)提供的图形界面调试器以及 SDT2.51 都与 Multi-ICE DLL 兼容，但由第三方提供的调试器必须遵循 ARM RDI 1.5.1 的标准才可使用。

如果你在 UNIX 下运行一个调试器,比如 AXD,则必须使用另一台可以运行 Multi-ICE server 的电脑与之相连。运行 UNIX 的工作站则必须安装支持到 Multi-ICE server 的 TCP/IP 连接的网络软件，并且至少要达到调试器安装所需的最低配置标准。

网络软件

如果你需要远程访问 server 程序，则操作系统必须支持网络软件。如果在安装过程中没有添加 TCP/IP 协议，就会弹出一条警告信息：

TCP/IP protocol does not appear to have been set up on this computer. Setup will continue. Please install TCP/IP if you want to use Multi-ICE remote access features.

即'此安装过程中没有加载 TCP/IP 协议。但安装过程仍将继续执行。如果你想使用 Smart-ICE 的远程访问功能，请安装 TCP/IP 协议。”

自动拨号

当你使用 Smart-ICE 时可能会触发自动拨号连接，因为 Smart-ICE 具备相应的网络功能。你可以使用下列任一方法撤消不必要的拨号：

在 Multi-ICE server **Settings**（设置）菜单上禁用 **Allow Network Connections**（允许网络连接）功能，只在 DLL 中使用本机作为服务器名；

禁用自动拨号功能。

1.1.2 主机硬件要求

此节确定了在 Windows 中安装并运行 Multi-ICE DLL 与 Multi-ICE server 的最低硬件要求。

硬盘空间

如果要完全安装这些软件，需要接近 20MB 的硬盘空间。

在 Windows 中运行 Smart-ICE 软件

要在 Windows 中使用 Multi-ICE server 和 DLL，你需要准备以下硬件：

200MHz Pentium 处理器；

系统存储器空间：

— 32MB RAM for Windows 95, Windows 98, and Windows Me

— 64MB RAM for Windows NT 4.0 and Windows 2000.

CD-ROM（也可以使用网络共享 CD-ROM）；

一个系统支持的 VGA 或更好显示方案的图形设备；

并口；

网卡（当需要远程访问 server 时）。

在 Solaris 上运行 Multi-ICE DLL

要在 Solaris 上运行 Multi-ICE DLL，你需要以下硬件：

Sun UltraSparc 或兼容机；

带有 *Common Desktop Environment*（CDE——通用桌面环境）的 Solaris 2.6，7.0 或 8.0；

CD-ROM（也可以使用网络共享 CD-ROM）；

可用的网络接口。

在 HP-UX 上运行 Multi-ICE DLL

要在 HP-UX 上运行 Multi-ICE DLL，你需要以下硬件：

HP PA-RISC v1.1 或 v2.0 处理器；

HP UX 10.20 或 11；

CD-ROM（也可以使用网络共享 CD-ROM）；

可用的网络接口。

在 Linux 上运行 Multi-ICE DLL

200MHz Pentium 处理器；

Redhat Linux 6.2 或 7.1；

CD-ROM（也可以使用网络共享 CD-ROM）；

可用的网络接口。

1.1.3 目标硬件要求

Smart-ICE 使用起来相当灵活，但也有相应的目标硬件要求：

遵循 IEEE1149.1（JTAG）标准的设备接口；

接口所接收的的电子信号，其电流和电压应在第 4 部分 *System Design Guidelines*（系统设计指导）中所要求的范围内；

目标板上的 IDC 接口应按 Smart-ICE JTAG 接口连接的要求布线，除非你使用老式的 14 位的 EmbeddedICE 接口或自制的新线；

目标板和 Smart-ICE 仿真器之间的最大线长为 20cm。

如果要支持多个 ARM 内核的 CPU,需要将 JTAG 扫描链调试逻辑串连。主要指的是 ARM7 和 ARM9 芯片，ARM1020T 芯片以及 Intel Xsacle 微处理芯片。但不包括 StrongARM®芯片。

所有支持的芯片，包括所有的 CPU，都列在安装目录\docs\proclist.txt 文件中了。

1.2 连接 Smart-ICE 硬件

此节讲述了如何连接 Smart-ICE 的硬件设备：

你需要的；

连接指导。

1.2.1 你需要准备的

要在 Smart-ICE 的硬件组件中选定你需要的，请参看图 1：

并口数据线，一公一母 25 芯并口连接线；

JTAG 线，一条两端都有一个方形的绝缘移位接头（IDC）插座；

Smart-ICE 仿真器主机。

取决于所使用的目标硬件的类型，你也许需要以下组件：

供电器，凭借一个 2.1mm 的插头，可以给 Smart-ICE 仿真器提供 5V 的电压。从而避免靠目标板给 Smart-ICE 供电；

JTAG 接口转接器。它是一个小的 PCB 板，上面有一个 14 针和一个 20 针的插座。用于采用 14 针标准的目标板

同时你还必须提供以下组件：

带有一个可用的并口的 Windows 操作系统计算机，用以运行 Smart-ICE Server 软件；

带有 Smart-ICE 支持的 ARM 内核 CPU,JTAG 接口的目标板。

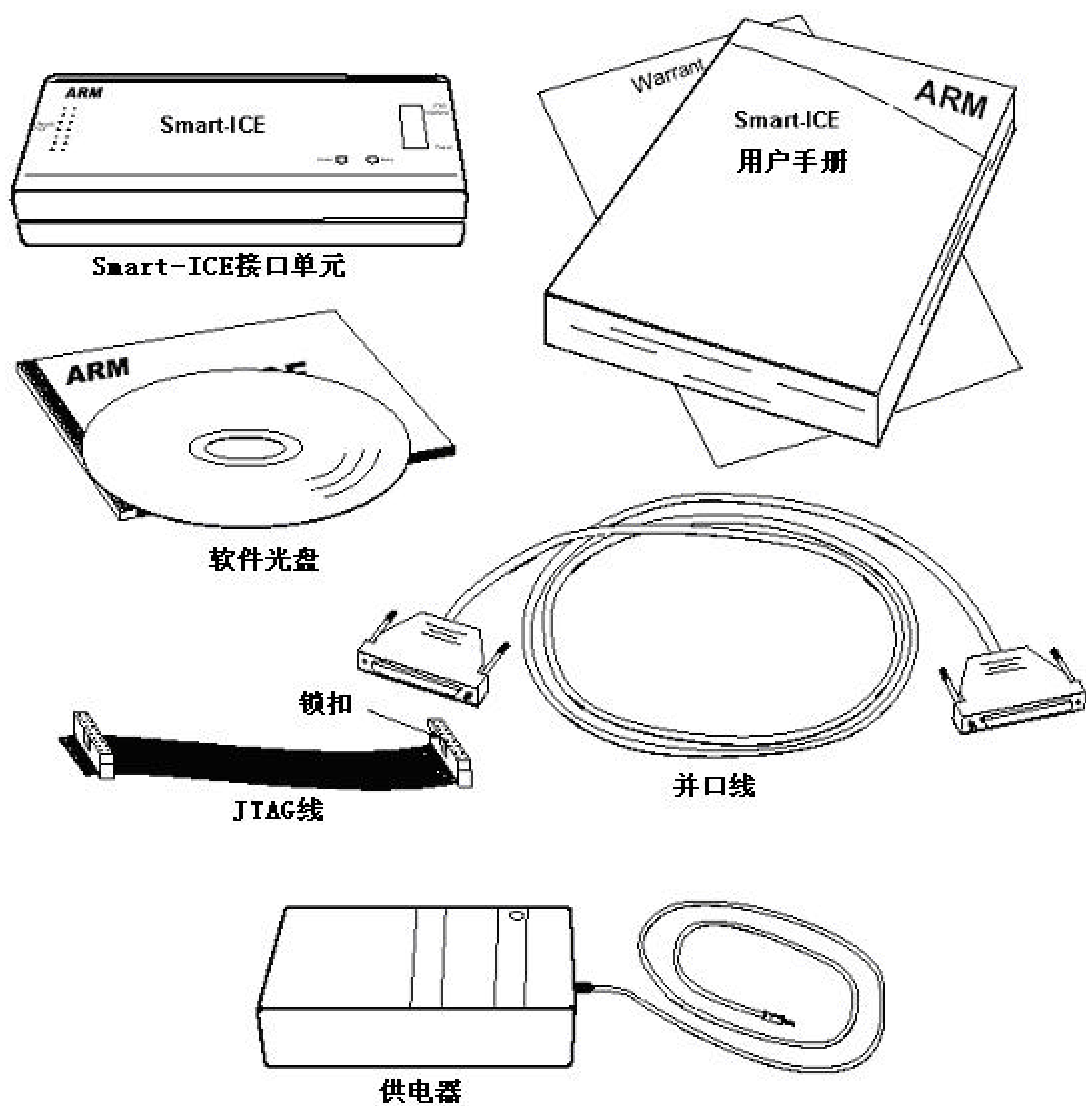


图 1　Smart-ICE 的硬件组件

1.2.2 连接指导

接下来你必须将 Smart-ICE 与你的工作站以及目标硬件连接起来。图 2 展示了这样一个例子：

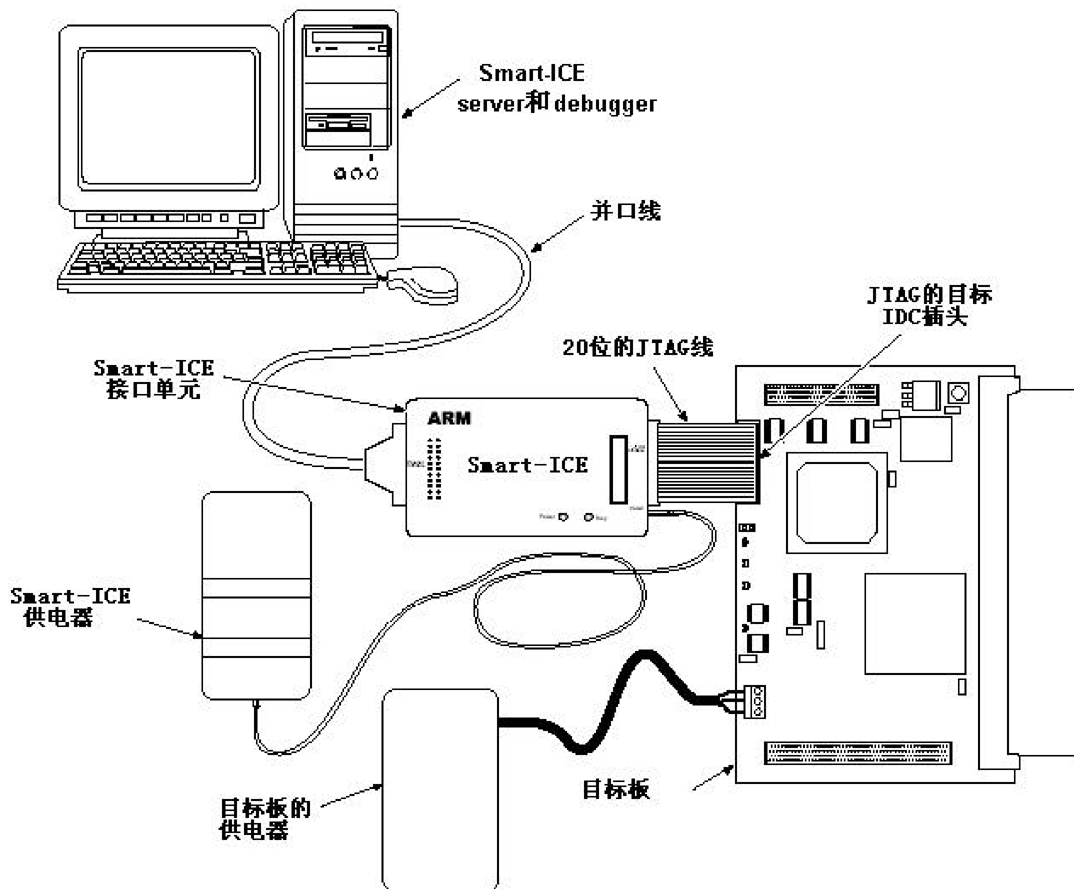


图 2 Smart-ICE 的组件连接示意图

要将硬件连接到一起：

1 保证在主机上已经安装了 Smart-ICE 的相关软件；

2 通过一条并口线，将主机的一个并口（可能标为打印机，并口，IEEE1284 或图形接口等）与 Smart-ICE 相连；

3 保证 Smart-ICE 有一个电源给它供电：SmartICE 可以使用两种方式供电，使用外部电源单独供电或者从目标板直接取电。如果您确认系统可以支持足够的电力（考虑到有些用户使用电池供电），可以不使用外接电源，将 ICE 通过 JTAG 连接，SmartICE 可以正常工作（可以看到电源灯亮）。

4 JTAG 的连接插头有小块突起以配合插座上的凹槽保证插接方向的正确。

5 如果还没有给它加电，则打开目标板上的电源。

1.3 连接非标准硬件

此节描述了当目标板上没有 ARM 的 20 位的 IDC 接头时，如何组建 Smart-ICE。共包含以下几部分：

非标准的接头；

电源供应。

1.3.1 非标准的接头

Smart-ICE 配备有符合 ARM 标准的 20 位 IDC 插座。目前所有的 ARM 开发板以及一些第三

方目标板上都装有与之匹配的插件。一些 ARM 开发板，比如 ARM 公司 PID 开发板，使用的是 14 位的插座，在数据通信上与新型的 20 位插座兼容。通过本产品套装中的 20 转 14 芯转接头，可以把仿真器与这些目标板连接起来。

1.3.2 电源供应

Smart-ICE 仿真器通过以下器件得到电源供应：

20 位的 JTAG 接头的 pin2（Vsupply），从目标板电源处获取电流供应；

电源输入插座

目标板电源最低电压 2V，最高 5V。你可以通过下列公式计算出大概的工作电流：

$$800mA \times \left(\frac{1.5V}{targetV}\right)^2$$

图 4 显示了此公式的函数图像。在加电过程中，Smart-ICE 仿真器将获取比图中所示更多的电流，从而也要求供电器必须符合此要求。按照常规情况计，3.3V 时电流应该为 180mA。如果目标板的供电器提供的电压或电流过低，就必须使用外部的电源输入插座。

你可以使用套件中提供的电源来给 Smart-ICE 供电。

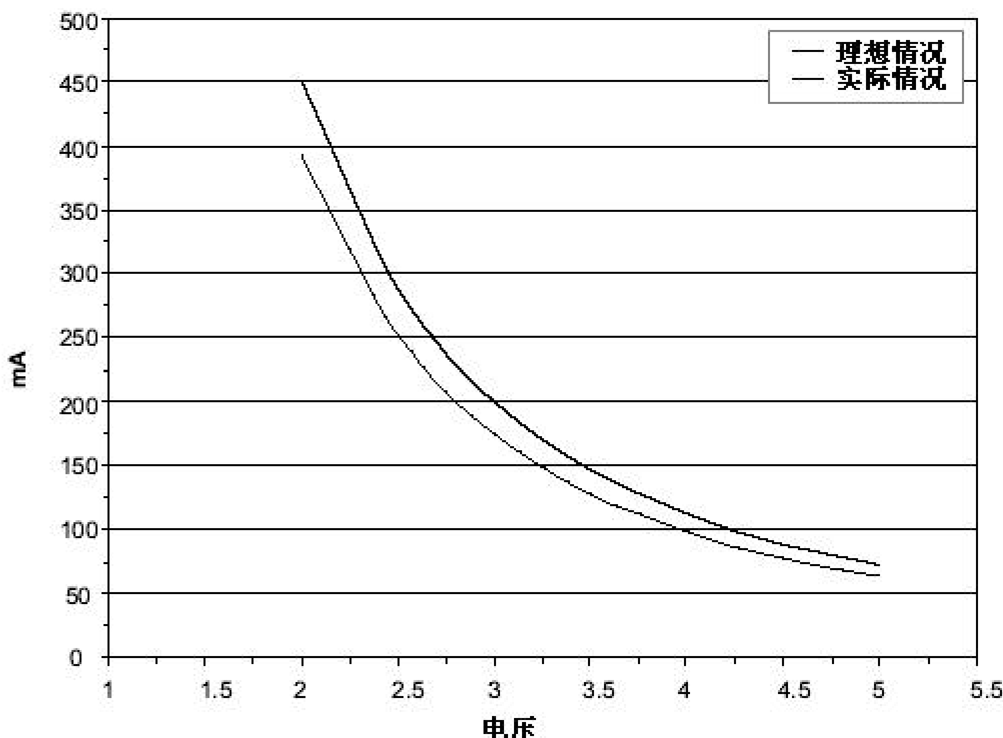


图 4 Smart-ICE 的电流随电压变化的衰减图

1.4 安装并启动软件

* 此节讲解了如何运行使用 Smart-ICE 的相关软件。

* 共包括以下几部分：

* Multi-ICE 的 *Microsoft Windows* 的启动程序菜单；

* 启动 Multi-ICE server；

* 其他的 Multi-ICE server 的启动特点。

1.4.1 Multi-ICE 的 *Microsoft Windows* 的启动程序菜单

在 *Microsoft Windows* 系统的计算机上安装好 Multi-ICE,其窗口程序菜单的各项如图 5 所示。

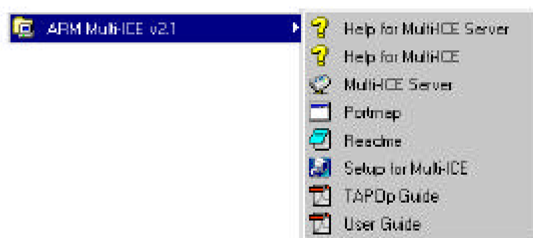


图 5　Multi-ICE 的启动菜单项目

其具体项目为：

Multi-ICE server 的帮助文件

　　显示 Multi-ICE server 的在线帮助文件；

Multi-ICE 的帮助文件

　　显示 Multi-ICE 的在线帮助文件；

Multi-ICE server

　　运行 Multi-ICE server；

端口映射器

　　运行端口映射器，要求从网络上访问 server；

自述文件

　　在 Windows 写字板程序中打开产品的自述文件。它包含了一些额外的设备信息。

设置 Multi-ICE

　　运行 Multi-ICE 设置程序。它允许你安装额外的组件，修复或删除 Multi-ICE 软件。当你执行删除或修复操作时，已经移动过或在安装 Multi-ICE 过程中生成的文件不会被删除。

TAPOp 手册

　　使用 PDF 文件浏览器显示 Multi-ICE TAPOp API 参考手册的内容；

用户手册

　　使用 PDF 文件浏览器显示 Multi-ICE 的用户手册的内容。

注意：上述两项，必须预先安装支持 PDF 文档浏览的软件。

1.4.2 启动 Multi-ICE server

要启动 Multi-ICE server，必须做到：

1 保证：

Smart-ICE 已经连接到工作站上；

Smart-ICE 已经与目标板的 JTAG 接口相连；

目标板已经打开电源或连接好电源；

Smart-ICE 的红色电源信号灯已经亮起（READY 的信号灯会闪动表示准备完毕）。

2 选择 **Start→Program→ARM Multi-ICE v2.2→Multi-ICE server**

显示 Multi-ICE server 窗口，如图 6 所示。端口映射器程序也可以同时将启动，并自动最小化，只要在主机上进行相关配置即可。

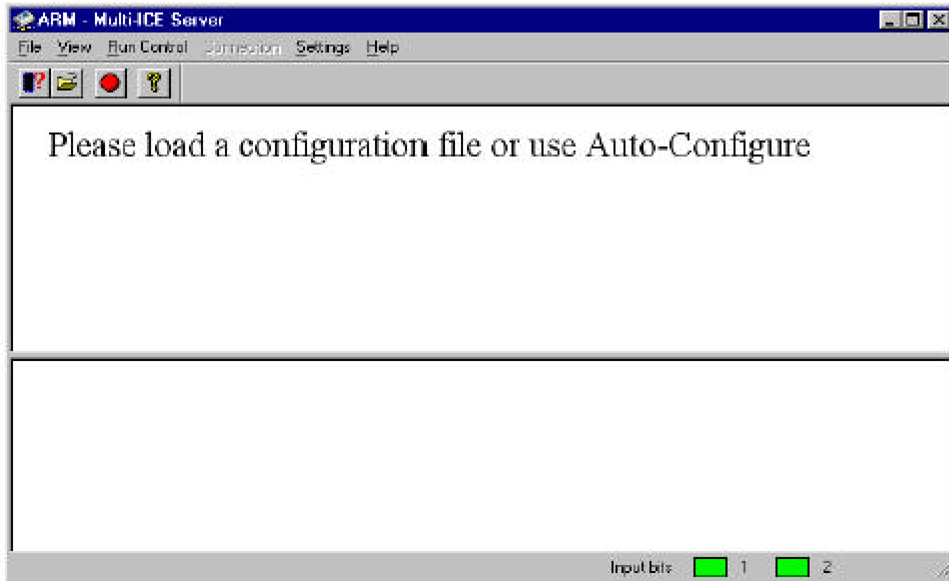


图 6　未配置的 Multi-ICE server **窗口**

3 **如果弹出一条找不到** Smart-ICE **硬件的信息的话，点击** OK**，然后再检查一下第** 1 **步中的各项；**

4 **配置** server**。通常可以通过自动配置命令来完成。选择 File→Auto-configure,等待直到** server **检测到目标板。如果** server **提示设备为"未知",则必须手动配置** server**。配置成功后，屏幕将如图** 7 **所示。**

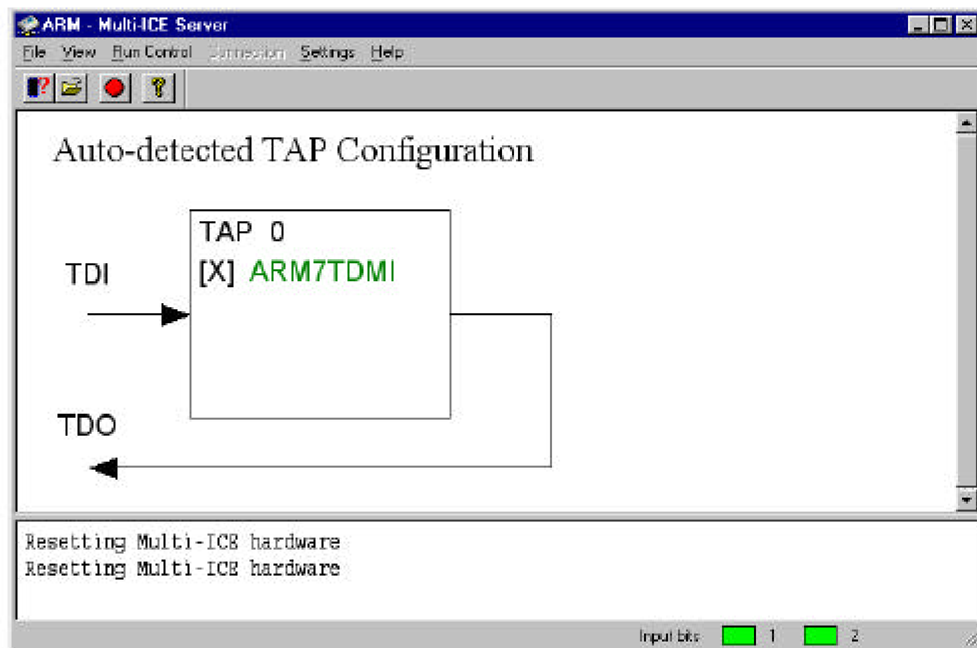


图 7　ARM7TDMI **的** Multi-ICE server **配置窗口**

1.4.3 其他的 Multi-ICE server 的启动特点

此节讲述了更多关于启动 Multi-ICE server **的信息。**

在没有安装 TCP/IP 协议的工作站上运行 server

如果在工作站上没有加载 TCP/IP **协议，当你第一次启动** Multi-ICE server **时，就会弹出一条**

警告信息，并且 Settings （设置）选项“Allow Network Connections”（允许网络连接）将自动变为不可访问状态。
直到进入调试阶段时，网络设定功能才被启用。

不连接目标硬件的运行
启动 Multi-ICE server 软件，而不与 SmartICE 相连。将弹出以下信息对话框：

```
Could not find the Multi-ICE hardware. Please check that the hardware is properly
connected to the parallel port and powered up
```

即“找不到 Multi-ICE 硬件。请查看目标硬件是否正确地连接到并口，并打开了电源。”
这条信息只是一个警告。要正确运行 server：
1 保证仿真器与并口以及目标板相连，并且打开了电源；
2 使用 File→ Load configuration（加载配置方案）或者 File→ Auto-configure （自动配置）命令对 server 进行配置。

没有网络连接
有时，机器上没有安装网络软件，或者网络设置出错，Multi-ICE server 在启动完成后都会立即中止运行。这意味着你没有机会取消“Allow Network Connections（允许网络连接）”这一设置。
如果你碰到这一问题，在 Windows 浏览器汇中运行 Non_tcp_ip.reg 脚本程序(位于 Multi-ICE 的安装目录下)。它将自动在系统注册过程中取消允许网络连接的设置，从而避免 server 尝试连接到网络。

最小化启动 server
你可以创建一个快捷方式图标，使 server 启动时最小化。按以下步骤完成：
1 在 server 图标上点击右键，选择 Create Shortcut（创建快捷方式）；
2 在快捷方式图标上点击右键，选择 Properties（特性）；
3 点击 Shortcut（快捷方式）标签；
4 在 Run 的下拉菜单中选择 Minimized 选项；
5 点击 OK。
最后，要最小化启动 server，双击该快捷方式即可。

Part 2. 运行 Multi-ICE server

这一章介绍了如何运行 Multi-ICE。它包含以下几部分：
* Multi-ICE 菜单的相关内容；
* Multi-ICE server 的设备配置文件；
* Server 的配置；
* 在多处理器模式下运行 Multi-ICE server。

2.1 Multi-ICE 菜单的相关内容

这一节简单介绍了 Multi-ICE server 的菜单项：
* 菜单结构；
* 文件菜单；
* 查看菜单；

* **运行控制菜单；**

* **连接菜单；**

* **设置菜单；**

* **帮助菜单。**

2.1.1 菜单结构

图 8 **显示了相关子菜单和其相关菜单项。**

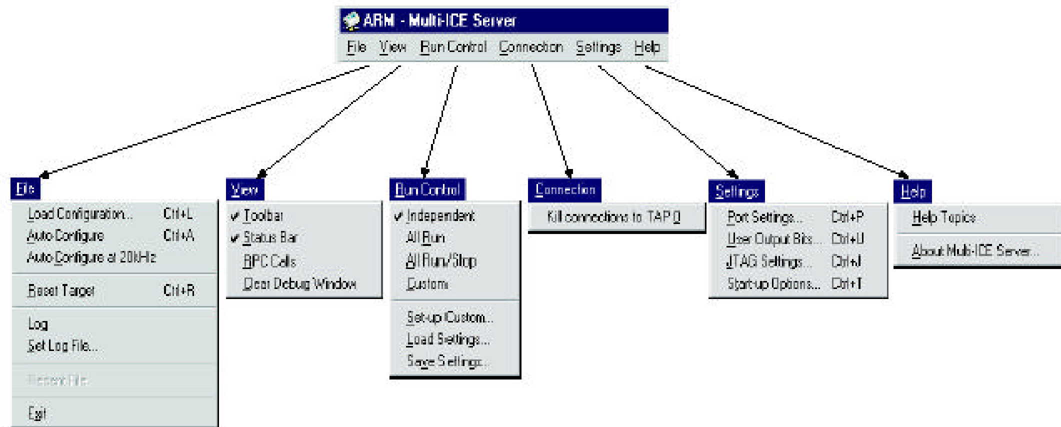


图 8　Multi-ICE server **的菜单项**

2.1.2 文件菜单

文件菜单允许你配置 Multi-ICE server**，并控制** TAPOp **请求的记录。如图** 9 **所示。**



图 9　**文件菜单**

此菜单包含以下菜单项：

Load Configuration（加载配置方案）

显示一个对话框，你可以在其中输入配置文件的名称和路径。此项用以手动配置 Multi-ICE**。具体内容在** Irlength.arm **配置文件中有详细说明；**

Auto-Configure（自动配置）

查询连到 JTAG **扫描链上的设备，并创建一个配置文件，包含查询结果的信息。表** 2 **列出了实际使用的频率。**

表 2 **自动配置的** TCK **频率**

Menu entry	自动配置过程中的 TCK频率	普通操作过程中的 TCK频率
Auto-Configure	1MHz	10MHz[a]
Auto-Configure at 20kHz	20kHz	20kHz

a. 1MHz 是针对带有不止一个 TAP 的目标板，或者是那些需要降低频率的设备

未知设备被标注为 UNKNOWN，但你可以将它们添加到查看列表中。
注意：如果一个已知处理器在自动配置时失败（则显示为 UNKNOWN），按硬件复位键，或关闭电源后重新加电，再次尝试自动配置。

Auto-Configuration at 20kHz（在 20kHz 频率处执行自动配置）

此项与 Auto-Configure 作用一样，但其使用的 TCK 频率绝对不超过 20kHz。当 JTAG 线或相关设备不能保证在较高频率时正常工作，或当设备处于休眠模式时，就应该使用本选项（当系统时钟减慢，而设备无法快速地对较高 TCK 作出响应）。

当使用了手动配置模式，就将使用 **Auto-Configure** 对应的 TCK 频率，除非配置文件指定了其他的时间参数。

Reset Target（复位目标板）

复位目标硬件。点击工具栏上的 **Reset Target** 按钮，相当于在菜单中选取该命令。

你可以控制 **Reset Target** 的动作，因此它可以使用 **nSRST** 或 **nTRST**，甚至同时使用这两种信号。你可以在 **JTAG Settings**（JTAG 设置）对话框进行上述设置，或者在配置文件中的 Reset 段进行相关设定。

你可以从调试器复位目标硬件，只要使用 system_reset 中间参数。此公式是对 **nSRST**，不是 **nTRST**。

Log（记录）

启用或关闭远程进程调用的记录功能。当启用时，该菜单项旁将显示一个小勾图形，同时 server 收到的 TAPOp 协议请求的文本内容也将记录到一个日志文件中。

使用 **Set Log File** 项可以指定使用的文件名。

Set Log File…

显示一个对话框，可以在其中输入该日志文件的名称和路径。

Recent File

显示最近使用过的 8 个配置文件。

Exit

关闭 Multi-ICE server。

2.1.3 查看菜单

View（查看）菜单控制 Multi-ICE server 窗口的显示，如图 10 所示：

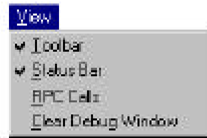


图 10　　查看菜单

此菜单含有下列菜单项：

Toolbar（工具栏）　　打开或关闭工具栏。当屏幕上显示了工具栏，则在该菜单项旁边将出现一个小勾。工具栏选项使你能快速调用以下函数：

File → Auto-Configure 尝试自动配置 server

File → Load Configuration 提示你输入配置文件的名称

File → Reset Target 使用 nSRST 或 nTRST 或二者同时使用，从而使目标硬件复位

Help → Help Topics 显示 Multi-ICE 的在线帮助。

Status Bar（状态栏）　　打开或关闭状态栏。当屏幕上显示了状态栏，则在该菜单项旁边将出现一个小勾。它显示了 Multi-ICE 的当前状态信息。

RPC Calls（RPC 调用）　　打开或关闭调试窗口。当激活了调试窗口后，该菜单项旁边将出现一个小勾,并且所有的 TAPOp 请求信息都将显示在调试窗口中。

Clear Debug Window（清空调试窗口）　　清除调试窗口中的所有文本内容。

2.1.4 运行控制菜单

Run Control（运行控制）菜单控制多芯片的同步中止和启动。如图 11 所示。



图 11　运行控制菜单

该菜单含有以下菜单项：

Independent（独立）　　使所有已配置过的设备独立运行。设备间没有交互操作。

All Run（全部运行）　　使所有设备尽可能地同步运行起来。

All Run/Stop（全部运行/中止）　　使所有已配置过的设备：

	当所有已连接的调试器都指示处理器可以开始运行时，使全部设备开始运行； 如果任何一个设备停下来，所有设备也都停止运行。
Custom（用户自定义）	使所有已配置过的设备进行交互操作，只要你指定了它们的 Set-up Custom 行动。
Set-up Custom…（设置自定义）	显示一个对话框，可以在其中指定设备交互操作的方式。
Load Settings …（加载配置项）	显示一个对话框，可以通过它加载之前保存过的运行控制的相关配置项。
Save Settings …（保存配置项）	显示一个对话框，可以将运行控制选项保存为一个文件。

2.1.5 连接菜单

Connection（连接）菜单列出了所有使用中的 TAP 控制器，并可以分别地取消这些连接。如图 12 所示。



图 12　连接菜单

警告：除了取消连接操作外，其他尝试从客户端断开连接的操作都是无效的。将 server 和一个活跃的客户端连接取消后，有可能造成客户端冲突或导致意外结果。

2.1.6 设置菜单

Settings（设置）菜单允许你配置 server 使用 JTAG 端口的方式。如图 13 所示。

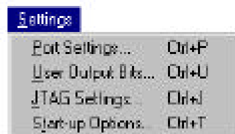


图 13　设置菜单

此菜单包含以下菜单项：

Port Settings…（端口设置）

显示一个对话框，你可以在中间选择需要的并口地址，并带有 4 位地址访问选项。它还显示了当前的端口模式。

User Output Bits…（用户输出位）(注:Smart-ICE 暂时不提供该功能)

显示了控制用户输出位的对话框。这些位数对应着用户 Input/Output（I/O）接头的两个逻辑输出信息。你可以使用这些信号来远程控制 server 地址中的用户逻辑信息。

JTAG Settings…（JTAG 设置）

显示用来设置时钟速度的对话框。使用 Set Periods Manually（手动设置周期）选项，或者在一个配置文件中添加其相关信息，从而在预定义的频率中选定时钟速度。如果在配置文件中选定了时钟信息，它将自动被选定。

Start-up Options（启动选项）

显示一个对话框，用来指定 server 启动时所进行的操作。

2.1.7 帮助菜单

Help（帮助）菜单使你获得在线帮助和 Multi-ICE 的相关信息。如图 14 所示。

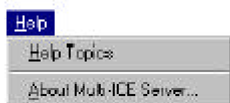


图 14　帮助菜单

此菜单包含以下菜单项：

Help Topics（帮助主题）

启动 Multi-ICE 帮助。

About Multi-ICE server…（关于 Multi-ICE server）

显示 Multi-ICE 的软件和硬件模块的版本信息。在获取供货商的技术支持时，你必须提供这些信息。

2.2 Multi-ICE server 的设备配置文件

Multi-ICE server 使用一个配置文件来存放板子上的设备的相关信息。Multi-ICE 需要配置数据来选择正确的驱动。在支持 ARM 的设备上都有这些配置数据。

创建一个配置文件可以分为以下两个方面：

Automatic device configuration（自动配置设备）；

Manual device configuration（手动配置设备）

2.2.1 Automatic device configuration（自动配置设备）

如果你的 ASIC 上的所有芯片都支持 ARM 芯片的话，Multi-ICE 可以通过扫描 ASIC 和创建文件 autoconfig.cfg 来创建配置文件。从菜单选择 **File→Auto-Configure**，或者选择 **File→Auto-Configure at 20KHz**（针对速度稍慢的设备），可以完成创建。

配置文件要包含每个 TAP 控制器的信息。Multi-ICE 包含了所有支持 JTAG 通信的 ARM 设备的必要信息。如果设备中没有使用 ARM 设备，则必须向 Multi-ICE 声明此情况，以使它们列入配置图表中。

要自动创建一个配置文件，选择 **File→Auto-Configure**。Multi-ICE 显示了探测到的设备的图形以及在扫描链中出现的顺序。如图 15，它显示了配置 ARM940T 的结果。

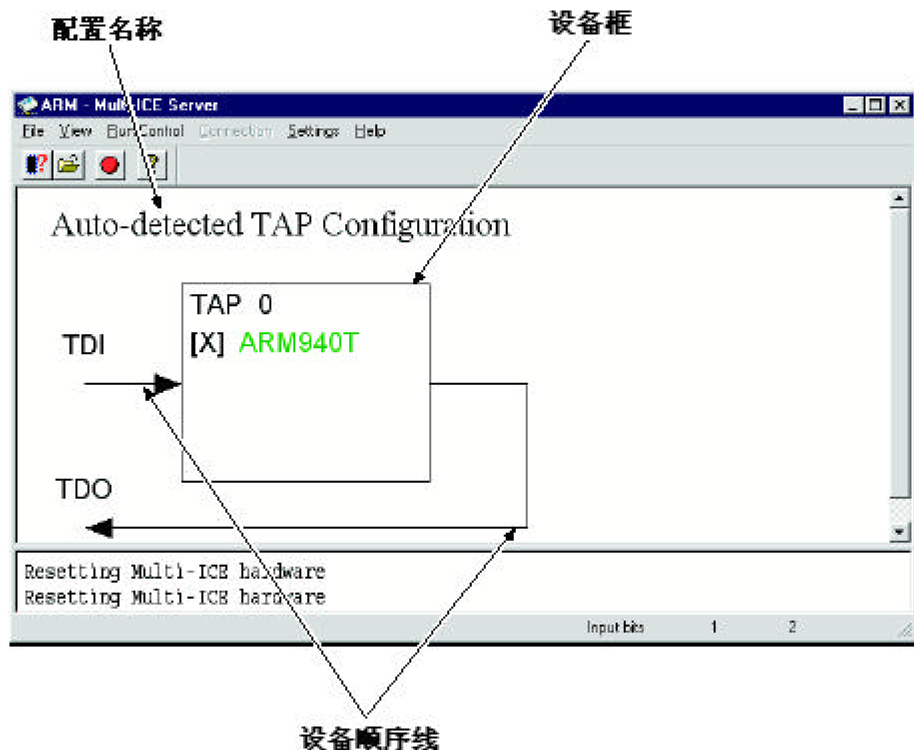


图 15 自动配置一个 ARM940T

注意 :" 自动配置 " 在确定配置时将执行几次系统复位。如果你的目标设备不允许这样做，则稍后你必须手动配置 server。

图 15 显示了：

该配置的名称。允许你对配置进行标记；

在扫描链中找到的设备的顺序。标有 TDI（Test Data In——测试数据输入）的箭头表示从 Smart-ICE 传送过来的 JTAG 数据，而标有 TDO（Test Data Out——测试数据输出）的箭头则表示进入 Smart-ICE 的数据；

每个方框对应一个已知设备。方框内的符号表示：

—每个芯片的状态符号：

[S] 芯片已停止运行；

[R] 芯片正在运行；

[D] 芯片正在加载程序；

[X] 芯片状态未知（没有已连接的调试器)。

—该设备的 TAP 数目；

—设备型号。该型号已列在 irlength.arm 文件中，并且不可被更改。

Multi-ICE 将配置信息自动写入到一个文件中，并且在需要使用它时自动加载。该文件放在 Multi-ICE 的安装目录下（比如，C:\Program Files\ARM\ARM Multi-ICE\），其名称为 autoconf.cfg。在图 15 中显示的配置信息对应的文件如例 1 所示。

例 1 ARM940T 的自动配置文件：

```
;Total IR length = 4

[TITLE]
Auto-detected TAP Configuration

[TAP 0]  ;IR_len=4, ID_code=1F0F0F0F
ARM940T

[Timing]
Adaptive=OFF
```

此文件包含了配置名称，TAP0 连接 ARM940T 的接口和一些时间数据。分号;后的内容是文件中的注释信息。

注意：

就像自动配置的一部分一样，只要复制和重命名自动生成的配置文件，你就可以创建一系列不同的 server 配置方案，而不需要改写原配置文件或要求 Smart-ICE 复位目标板。

2.2.2 Manual device configuration（手动配置设备）

手动配置 Multi-ICE server，也可以创建一个 server 配置文件，并将其记载到 Multi-ICE server 中。要创建该文件：

1 首先查看 proclist.txt 文件，确定 Smart-ICE 是否支持需配置的设备：

如果不支持该设备，你可以创建一个设备接口，告知 Smart-ICE 扫描链确定该设备占用的寄存器所需要的时间，从而达到命名该设备的目的；

如果支持该设备，则直接在文件 IRlength.arm 中选定设备接口的名称即可。

2 打开一个文本文件编辑器，创建一个新文件，使用.cfg 为文件扩展名。

3 至少在文件中键入以下内容：

TITLE（标题）部分；

TAP0 部分，包含 IRlength.arm 文件中第一设备的名称。

其他部分的内容由你所要配置的设备来决定。

4 保存文件。

5 在 Multi-ICE server 中，点击 **File→Load Configuration…**

6 确定你所创建的配置文件的路径，点击 **Open**。

7 Multi-ICE server 将按你的配置文件中的信息被配置完成。

注意：当你加载一个手动配置文件时，Multi-ICE server 将不会检测所配置的设备是否与实际的扫描链相匹配。你需要自己确认一下。

命名一个不支持设备

要命名一个单独的 UNKNOWN 设备：

1 你必须找出该未知设备中的 TAP 控制器的指令寄存器的长度。要完成这一点，只需：

a. 自动配置目标设备；

b. 在 Multi-ICE server 窗口中双击该 UNKNOWN 设备。此时将弹出一个包含 IR 长度的对话框。

2 在 Multi-ICE 安装目录下创建一个名为 USERDRVn.TXT 的文本文件（n 指的是该未知设备的 IR 长度）。该文件应该包含设备名称。

例如，要命名一个包含 DSP 的 TAP 控制器的目标设备，其 IR 长度为 4 位，你就需要创建

一个包含 DSP 名称的 USERDRV4.TXT 的文件。
注意：你不能以同一 IR 长度命名多个设备，因为 Multi-ICE server 不能区分它们。

IRlength.arm 配置文件
Multi-ICE server 通过每个 IR 寄存器在 JTAG 扫描链查到的长度来计算整个扫描链的长度。这些信息都被放在一个名为 IRlength.arm 的文件中。你可以编辑此文件，将相关信息存储到其他设备上，比如一个 DSP。此文件放在 Multi-ICE 的安装目录。IRlength.arm 文件的摘录信息如例 2 所示。
例 2　IRlength.arm 配置文件的摘录信息

```
;ARM7 series cores
ARM7TDMI=4
ARM7TDMI-S=4
ARM740T=4

;ARM9 series cores
ARM9TDMI=4
ARM920T=4
```

2.3 Server 配置

此节详细讲述了关于 Multi-ICE server 程序的配置：
芯片驱动设定对话；
启动选项对话；
并口设置对话；
用户输出字节数对话；
用户输入字节数对话；
JTAG 设置对话。

2.3.1 芯片驱动设定对话

如果你在如图 15 上的芯片的方形输入线上双击的话，Multi-ICE server 将给出更多关于所连接的设备的信息。读取设备的 TAP 控制器设定并将其按 server 中创建的标准列表方式“翻译”出来，就可以得到设备信息。该对话框如图 16 所示。
在对话框顶部有 **List of Drivers**（驱动列表），其中包含了 Multi-ICE 驱动的名称，而它们都与一个指定的 TAP 控制器相连。列表通常都含有一个主驱动（如图 15 中的 ARM940T），其 IR 长度值为正。它还可能包含其他的驱动，在 irlength.arm 文件中以 0 IR 长度值来标识。例如带有一个 ETM 的 ARM920T。
对话框中的 **Driver Details**（驱动详情）信息随每次连接不同而变化，甚至在 **List of Drivers**（驱动列表）的每个设备间都不同：

Connected To	所分配的连接的名称
At	此连接的创建时间
connectId	分配给此连接的连接数，在 RPC 日志文件中也可查看到
Vers. Reqd	创建连接时客户端要求查看的 server 协议的版本数

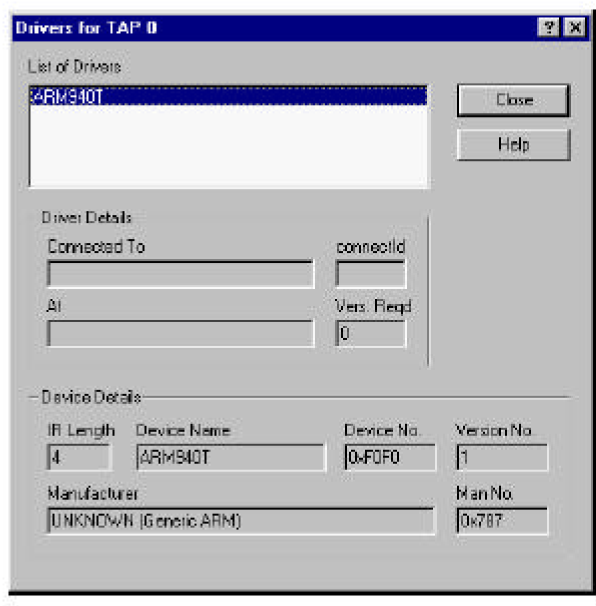


图 16　　TAP 驱动的状态对话框

在对话框的 Device Details（设备详情）处可以查看到设备的详细信息。不像 Device Name，当从 List of Drivers（驱动列表）中选择另外的驱动后，此详细信息不会更改，因为它是与设备相关联，而并非由驱动调用。

此对话框含有以下项目：

IR Length	指令寄存器的数位长度，是 JTAG TAP 控制器的首要元素
Device Name	设备的指定名称
Device No.	生产商指定的设备号码
Version No.	芯片的版本号
Manufacturer	生产商号码的文本版本
Man No.	JTAG 生产商代码

设备，版本和生产商号码都由 TAP 控制器的 ID 寄存器来读取。该寄存器中的值遵循以下格式，如图 17 所示。

31　28	27　12	11　1	0
Version	Part Number	Manufacturer identity	1

图 17 TAP 控制器设备 ID 寄存器的格式

注意：

根据 IEEE1149.1 关于设备确认的规则，Device No.域中的 0xF0F0 和 Man No.域中的 0x787 值不可用。

ARM 公司将这些值写入设备逻辑设计中，并提供给生产商，用以让特定设备的代码来覆盖。如果 ARM 的原始值没有被覆盖，Multi-ICE DLL 将使用 UNKNOWN（Generic ARM）串。设备名称和部件编号由设备特性来确定。如果设备不能够被自动配置，而只有采用手动配置时，Multi-ICE 就将显示真实的生产商编号和符号串。

2.3.2 启动选项对话

本节介绍的是启动选项的对话，如图 18 所示。

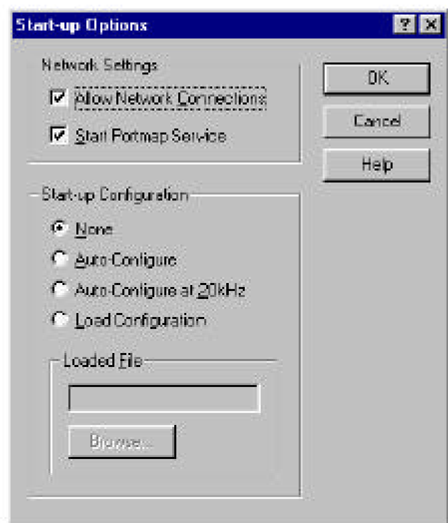


图 18　　启动选项对话

此对话框可以通过 Settings→Start-up Options…菜单调用，并被划分为两个组：

Network Settings（网络设置）

Start-up Configuration（启动配置）

Network Settings（网络设置）将启用或禁用 Multi-ICE server 的网络功能。其他的设置将控制初始的 JTAG 的设备配置。

对话框包含下列项目：

Allow Network Connections（允许网络连接）

选中此项，Multi-ICE server 就将允许其客户端通过 Sun RPC 协议远程访问网络。要求两边的工作站都必须安装 TCP/IP 协议。但是，为了达到以下目的，应不选此项：

避免与敏感设备进行网络连接；

允许一个没有安装 TCP/IP 协议的工作站启动后使用 server。

Start Portmap Service（启动端口映射服务）

如果你选择了此项，就将启动 Multi-ICE server 的端口映射服务。如果 Start Portmap Service（启动端口映射服务）窗口不可用，就必须在启动 Multi-ICE server 前通过其他方式启动端口映射服务（比如，选择 Start→Programs→ARM Multi-ICE v2.2→Portmap）。

注意：

如果没有选择自动启动端口映射服务的功能，并且没有其他的应用程序提供端口映射服务，则不会弹出警告信息。最后的结果是客户端不能连接到 server 上，并且一会儿将产生一个错误报告。

要解决这一问题，你必须再次启动 Multi-ICE server，选中 Start Portmap Service 选项，然后关闭 Multi-ICE server，最后再次启动 Multi-ICE server 即可。

当没有选中 Allow Network Connections 选项时，本项功能不能使用，因为此时并不需要端口映射服务。

None

如果选择了此项，则不执行任何的自动配置或加载配置文件的操作。当 server 启动后，其窗口如图 6 所示，并且在使用 server 之前必须进行相应配置。

Auto-Configure（自动配置）
此项将自动创建一个配置文件，用来给所有已探测到的设备命名。未知设备则标注为 UNKNOWN。

Auto-Configure at 20kHz（在 20kHz **执行自动配置）**
20kHz **的自动配置非常适用于系统时钟较慢的情况，比如说位于仿真环境中或处理器处于睡眠状态。**

Load Configuration（加载配置文件）
弹出一个用以输入配置文件名称和路径的对话框。此项用来手动配置 Smart-ICE**，并且在** IRlength.arm **配置文件中有详细描述。**

2.3.3 并口设置对话

此节介绍了如何进行 Port Settings（端口设置）对话。可以通过 Settings 菜单来调用此对话。

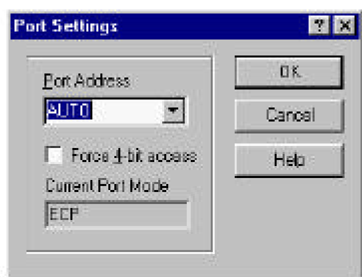


图 19　**端口设置对话**

此对话框包含以下选项：

Port Address
指的是所使用的并口地址。可以选择下列任意一个：

AUTO　　　　　**自动选择要使用的并口；**
LPT1　　　　　**选定** LPT1 **作为要使用的并口；**
LPT2　　　　　**选定** LPT2 **作为要使用的并口。**

注意：
你必须将 Smart-ICE **的接口连接到使用标准并口地址** 0x278 **或** 0x378 **的并行端口上。不能使用** USB **转并口线或者** PCI **接口的并口扩展卡。**

Force 4-bit access（强制使用 4 **位数据访问格式）**
要求并口强制使用 4 **位数据传输模式。**

Current Port Mode**（当前端口模式）**
此项指的是并口的工作模式。很多并口都有多种工作模式，取决于 BIOS **设置。你可以参看电脑硬件手册，**Smart-ICE **可以使用以下的并口模式：**

4-bit　　　　　**基本的单向并口模式；**
8-bit　　　　　8-bit **的双向并口模式；**
ECP　　　　　Enhance Capability Port**（增强端口）。此模式比** 8-bit **要快，因为它可以实现段数据传输。**

在某些主板，Smart-ICE **可能不能使用** ECP **模式，这个时候可以在** PC **启动的时候进入** BIOS**，把并口模式设置为** *Enhanced Parallel Port*（EPP—**增强并行端口）模式，或者双向的** 8-bit

模式。大部分情况下 Smart-ICE 可以在 ECP 兼容模式下使用 IEEE1284 端口。
注意：
Windows 95，Windows98 和 Windows Me 驱动不同使用 ECP 模式，因为具备 ECP 功能的并口驱动将干扰 Smart-ICE 的并口驱动的操作。
有些机器可能会因为不匹配的并口而导致随机性的通信失败。如果你发现很难连接上 Smart-ICE 硬件，或经常在操作过程中发生超时错误，就尝试一下 4-bit 的访问模式。

2.3.4 用户输出字节数对话(注:Smart-ICE 不提供该功能)

用户输出字节与两个 TTL 逻辑输出（位于用户的 I/O 接头）结果对应。你可以使用这些信号来远程控制 server 的用户逻辑关系。
你可以在 **Settings** 菜单中访问 **User Output Bits**（用户输出字节数）对话框，如图 20 所示。

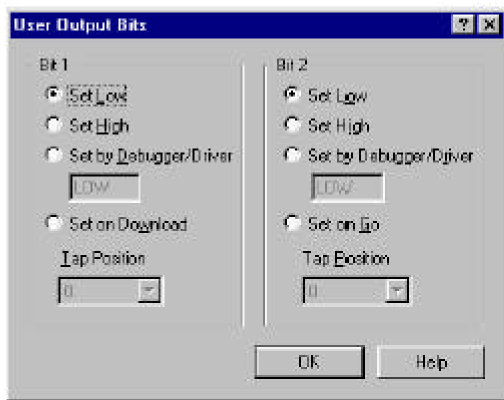


图 20　　用户输出字节数对话

注意：用户输出字节将随你的设定而即刻改变。你不需点击 **OK** 来让修改生效。
此对话框包含以下项目：

Set Low　　将字节数设置为 LOW

Set High　　将字节数设置为 HIGH

Set by Debugger/Driver
　　允许输出字节数由以下任一程序控制：
　　TAPOp 进程调用 TAPOp_WriteMICEUser1 和 TAPOp_WriteMICEUser2；
　　调试器的内部变量 output_bit_1 和 output_bit_2。

Set on Download
　　当调试器向指定 TAP 控制器下载时，将 bit1 设为 HIGH。

Set on Go
　　当调试器在指定的 TAP 控制器上运行映象文件时，将 bit2 设为 HIGH。

你可以选择 TAP 控制器以及在 **Tap Position** 下拉菜单中选定其用户输出字节数的数目。所有的 TAP 控制器都列在其中。

2.3.5 用户输入字节数（注:Smart-ICE 不提供该功能）

用户输入字节数与用户 I/O 接头的两个 TTL 逻辑输出对应。你可以使用这些信号来远程控制 server 的逻辑信号。
用户输入信号在 Multi-ICE server 窗口的右下角有显示，如图 21 所示。

绿灯亮时表示 HIGH；
绿灯灭时表示 LOW。

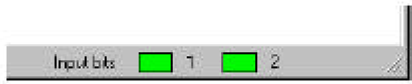


图 21　　用户输入状态

2.3.6 JTAG 设置对话

JTAG 设置对话框如图 22 所示，允许你选择 Smart-ICE 如何生成 JTAG 时钟和复位信号。直到目标硬件配置完成后，菜单方可用。你可以自动配置目标硬件或加载一个配置文件。

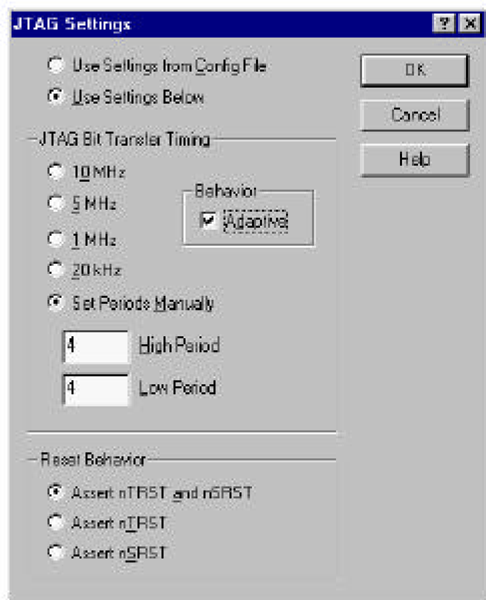


图 22　　JTAG 设置对话

在选定的目标配置文件中没有定义的设置将被自动设为默认值。使用此对话框可以：
使用选定的配置文件的时钟和复位信号；
定义你自己的时钟和复位信号。

要使用未修改的配置文件中定义的设定，点击 Use Settings from Config File（使用配置文件的设置）。
要在配置文件中修改已定义的设置，点击 Use Settings Below。执行此操作后将启用 JTAG Bit Transfer Timing 和 Reset Behavior 控制组。这些控制组的初始化设置由当前 server 的配置完成。
在 JTAG Bit Transfer Timing 组中，你可以定义基本的 TCK 频率，并选择恰当的时钟：
选定一个预定义的 TCK 频率：10MHz，5MHz，1MHz 或 20kHz。
选择 Set Periods Manually（手动设定周期），定义一个确定的 TCK 模块。
当目标设备必须使用合适的时钟时，选择 Adaptive（自适应）。
在 Reset Behavior 控制组中，你可以定义用以复位目标系统的信号。Smart-ICE 仿真器可以

控制两种信号：

nTRST　　　　　**当正确连接后，此信号只复位目标板上的 JTAG 逻辑关系；**

nSRST　　　　　**当正确连接后，此信号用来复位目标处理器和连接到外围设备，但不复位 JTAG 逻辑关系。**

当点击 File→Reset Target 或红色的复位工具栏按钮时，这些单选按钮允许你选定的组合。你必须参考目标硬件的说明书来确定关于这些信号的操作。

手动设置 TCK 周期

如果你选择了 Set Periods Manually（手动设置 TCK 周期），或者在配置文件中引入了时钟信息，则周期和频率将用下列公式来计算：

```
HIGH period  =  50ns * (high_scale * (high_multiplier + 1))
LOW period   =  50ns * (low_scale * (low_multiplier + 1))
Total period = HIGH period + LOW period

Frequency    = 1 / Total period
```

同样的 HIGH 和 LOW 周期值用在 server 配置文件中。

注意：

在一个很低的 JTAG 时钟处，server 所使用的并口驱动将占用大部分的工作站处理时间。从而使应用程序运行速度减慢。

你可以在 0－255 范围内设定 HIGH 和 LOW 周期。输入的 8-bit 值将被分为一个 3-bit 和一个 5-bit 值用来构成标度（S）和乘数（M）的值，如表 3 所示。

Scale			Multiplier				
7	6	5	4	3	2	1	0
S	S	S	M	M	M	M	M

表 3　**标度和乘数值**

乘数由低 5 位字节构成，允许值的范围为 0 到 31。表 4 显示了标度值。SSS 是三位位于字节高端的内容。Scale 是公式中所使用的值：

SSS	Scale
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

表 4　**确定时钟速度的标度值**

例 3 显示了如何对一些实例频率进行编码：

```
100kHz (approx)    HIGH = LOW = 162   [SSS = 5 (S = 32)      M = 2]
500kHz             HIGH = LOW = 19    [SSS = 0 (S = 1)       M = 19]
2MHz               HIGH = LOW = 4     [SSS = 0 (S = 1)       M = 4]
```

例3　JTAG HIGH 和 LOW 设置的派生值

自适应时钟

如果目标硬件提供 RTCK 信号，则选择 Adaptive clocking（自适应时钟）功能来使其时钟与芯片外部的处理器时钟同步。这将保证 JTAG 连接不产生同步异常问题。

注意：如果你使用了自适应时钟功能，传送延迟，门户延迟和同步要求将使时钟频率变得比不使用此功能时低。因此除非硬件设计要求使用此功能，否则不要采用自适应时钟。

2.4 在多处理器模式下运行 Multi-ICE server

如果目标硬件上有不止一个处理器，Smart-ICE 允许你将每个处理器连接到不同的调试器上。你可以完全独立地在每个处理器上调试运行的代码，设定中断点，下载映象文件或者启动/终止处理的运行，而不会影响到其他处理器。

要完成此设置，需要做以下几步：

1 确保 Multi-ICE server 显示了目标硬件上的所有芯片；

2 每个芯片都用你选定的调试器来运行一个实例测试程序；

3 对每个调试器使用 Multi-ICE DLL 配置对话框来选择不同的芯片。

注意：如果你使用的是ARM调试器，比如AXD，则可以使用调试器的load session功能，从而避免每次启动时都需要配置调试器的麻烦。

在多处理器模式下运行 Multi-ICE server 可分为以下几部分：

控制设备运行；

运行控制和调试器；

关于运行控制；

在设备间实现交互通信；

设定挂号通信频率。

2.4.1 控制设备运行

在多处理器系统中，处理器间交互通信，从而完成设计所要求的功能。Multi-ICE server 具备运行控制功能，允许你调试这些系统。运行控制功能允许你以配置方式同时启动或终止几个处理器的运行。使用此功能你可以：

将多个处理器设定在某个特定程序段和语句处，并同时启动它们。即 *synchronous starting*（同步启动）。Smart-ICE 保证这些处理器在一个 TCK 周期内几乎同时启动；

当某个处理器遇到中断点，可以强行让所有处理器都停止运行，从而当运行至中断点时你可以看到系统的集中效应。此操作被称为 *synchronous stopping*。所有的处理器几乎在同一时间停止运行（取决于它们对于停止请求的响应时间）。

你还可以设置更复杂的启动和中止条件。

2.4.2 运行控制和调试器

运行控制是 Multi-ICE server 的特性。因为 Multi-ICE server 有关于系统中的每个处理器的详细信息。在 server 中，你可以在不同处理器间配置并控制其交互通信操作。

注意：如果你要求控制程序的特定部分进行调试，最好先将 server 设置为 Independent（独立）运行控制，从而避免意外的程序中断发生。当程序运行到你关注的区域时，在 server 上设置好计划的运行控制选项，然后开始调试；

当你改变控制设置时，确认系统上的所有处理器都停止运行，因为当程序开始运行时这些设置将生效。

同步启动

如果你创建了一个同步启动组，但其中包含一个特殊处理器，当你启动此处理器时它并没有立即开始运行。调试器将显示一条信息，指示 server 在等待其他处理器一起启动。在所有的等待信息解除前，你必须使用其他的调试器去启动同步启动组中的每个处理器，从而使它们同步启动。

注意：如果你需要同步启动以及同步停止功能的话，可以在 server 中进行相关设置。

同步停止

当一个处理器停止运行（比如，因为一个中断点或查看点），与此处理器相连的调试器将显示处理器的状态。如果你通过 Multi-ICE server 将此处理器和其他的处理器都停止运行，则显示以下信息：

```
Server stopped the processor
```

现在可以检查该组中的其他处理器的状态。

2.4.3 关于运行控制

一个处理器可能因为很多原因而停止运行。包括以下情况：

该处理器相连的调试器处于“`halt processor`”(暂停处理器）模式；

在调试器上按下了停止按钮；

处理器触发了一个中断点或查看点；

处理器运行过一个矢量，并且允许捕捉矢量；

在处理器上运行的程序执行了一个 semihosting 调用，并且在调试器中可以选择标准的 semihosting；

调试器步进到下一条指令时；

外围硬件，比如 ETM，在芯片中使用了 DEGRQ 信号。

以上所有的这些中断运行都能被 server 检测到，并根据你的运行控制设置进行处理。当你使用 Independent 运行控制选项之外的设置时，其他的处理器就有可能一起停止运行。有以下几种情况：

当执行 semihosting 调用时，如果需要多芯片设备的运行控制，则在 AXD 的处理器属性对话框中必须设置 DCC Semihosting 选项。在 ADW 中，调试器的内部变量`semihosting_enabled` 必须设为 2；

调试器根据所需使用中断点，但应该与 Multi-ICE server 中使用的中断点一样。比如，如果你在 AXD 中创建了一个计数值为 50 的中断点，处理器每次运行到此中断点时都停止运行，直到第 50 次触发它时再继续运行。其具体过程如下：

每次处理器停止运行，server 将执行你指定的动作。但有可能与你预期的不尽相同；

每次处理器重新启动（在此例中为 49 次），运行控制设置将生效。如果你设置了同步启动和同步停止，则当正调试中的处理器碰到中断点时，所有此设置内的处理器都将停止运行。当调试器尝试重新启动这些处理器时，Multi-ICE server 将一直等到所有同步停止的处理器都运行起来位置。
当你在调试器中点击 **Step**，处理器就会执行停止动作。因为 step 是在下一条指令处设置一个中断点，然后再重新启动处理器的命令。当处理器触发了中断点，server 将停止运行其他的处理器。

当你使用调试器加载一个映象文件时，中断点将自动设置在 main 函数处。点击 **Go**，将执行初始化代码直到这个点，然后你必须再次点击 **Go** 以恢复程序运行。此中断点与其他中断点一样。

2.4.4 启动设备间的交互通信

运行控制对话的设备间交互通信表如图 23 所示。

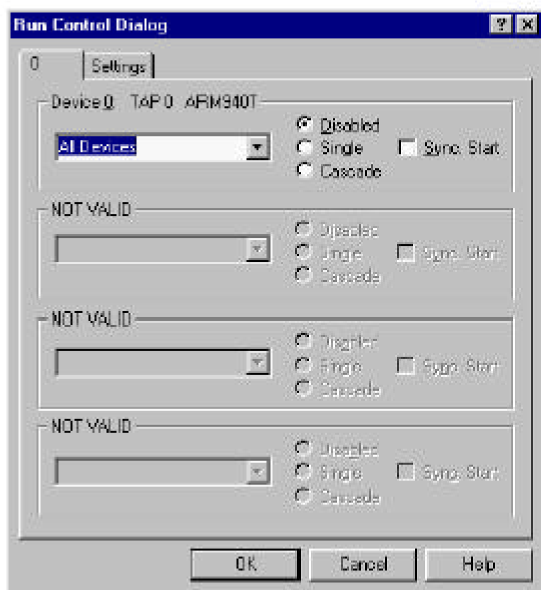


图 23　　　　设备间的交互通信

设备的 TAP 控制器数有多个标签，并按每段 4 个排列出来。这些标签使你可以在单个设备或一组设备间设置交互通信。
当所列设备少于四个时，有些对话框则不可用，并标记为 Not Valid。
每个列出来的设备都有一系列控制设置：
Range field 它是每个设备描述下方的下拉列表对话框。它允许你选择被当前设备停用的任何设备。默认设置为 **All Devices**。
如果有不止一个设备可用，你可以选择所有设备，单个设备号码或一组设备。
比如，你有 10 个列出的设备，可以使用下列表达来暂停 2，5，7，8 和 9 号设备的运行：
2，5，7-9

Disabled 取消对此设备的暂停运行设置，并且撤消对其他设备的控制；
Single 使 range field 中的被选中的运行设备即刻停止运行。如果以这种方式停止此设备的运行，则不会影响到其他设备；
Cascade 使 range field 中的所有运行中的设备即刻停止运行。如果以这种方式停止此设备

的运行，则其他设备也同时停止运行。如果一个设备本身没有运行，则忽略对其的停止运行命令。

注意：停止运行指令不对已停止的设备生效。

Sync.Start　　使用此选项，则将此处理器列为同步启动组的部分。当同步启动组中的一个处理器启动时，其他的处理器也同时启动。

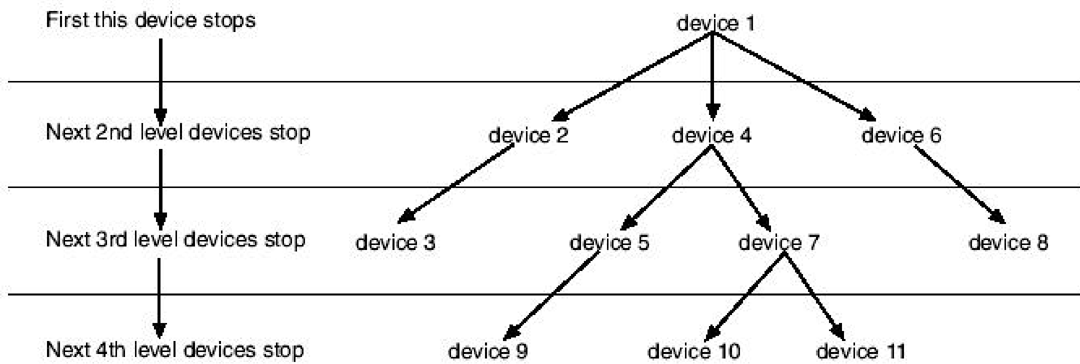


图 24　　Cascade 操作

联合设置

要达到更好的控制效果，你可以进行联合设置。如上图（图 24）所示：

如果设备 4 被禁用，则根据设备 1 的控制设置，设备 5，7，9，10 和 11 不会受影响；

如果设备 4 和 5 被设为 Cascade，设备 7 被禁用，则根据设备 1 的控制设置，将在设备 9 上生效，但不包括设备 10 或 11；

如果设备 6 被设置为 Single 从而取代 Cascade，设备 8 根据设备 1 的控制设置将不受影响。

2.4.5 设定查询频率

在 Run Control 对话框中的 Settings 标签如图 25 所示。

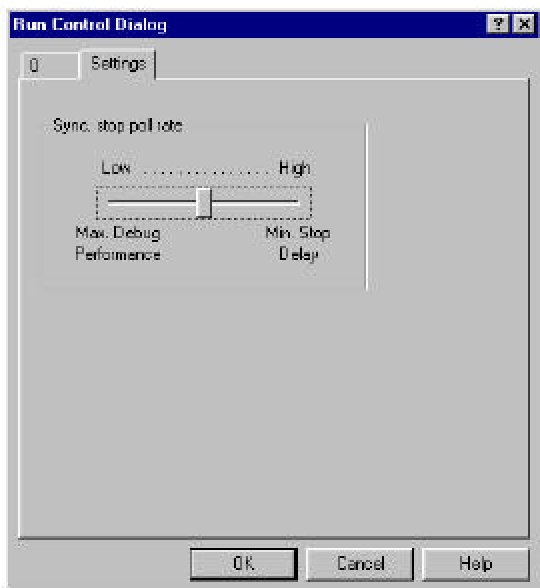


图 25 设定列举频率

它允许你修改 Smart-ICE 使用的设备查询频率。你可以在主机响应和 Smart-ICE 查询连接设

备的次数之间寻找平衡，从而确定其状态。
处理器进入调试状态，继而 Smart-ICE 探测到此状态，两步间存在一个延迟。其设置为：
Low 良好的调试器响应，最少的查询连接。如果处理器停止运行，不会马上探测到；
High 最快的查询连接，但调试器响应很差。如果处理器停止运行，将马上探测到。
默认设置为中间位置。

Part 3. Smart-ICE 的调试

本章将介绍如何进行支持 ARM 的调试器与 Multi-ICE DLL 之间的连接，并详细介绍了 Multi-ICE DLL 的特点。
共包含以下章节：
* ARM 调试器的兼容性；
* 将 Smart-ICE 连接到 ADW，ADU 或 AXD 上；
* 配置 Multi-ICE DLL；
* 配置并调试多处理器；
* Post-mortem 调试；
* 访问 CP15；
* Semihosting；
* 查看点和中断点；
* 缓冲数据；
* ROM 中的调试程序；
* 直接访问 EmbeddedICE 的逻辑结构。

3.1 ARM 调试器的兼容性

要调试你的 ARM 目标映象，可以使用以下的 ARM 调试器：

ARM eXtended Debugger (AXD) for Windows or UNIX
ARM Debugger for Windows (ADW) (SDT 2.51, ADS v1.0.1, or ADS v1.1)
ARM Debugger for UNIX (ADU) (SDT 2.51, ADS v1.0.1, or ADS v1.1).

现在新增加 arm 公司最新产品 RVDS 支持，你可以使用 RVDS
调试器与 Smart-ICE 配合一起工作。Smart-ICE 可以访问目标硬件，并提供正确配置调试器来访问目标硬件的工具。它提供的用户接口单元如寄存器窗口和分步调试器，十分有利于你的程序的调试。

3.2 将 Smart-ICE 连接到 ADW，ADU 或 AXD 上

手动加载一个配置文件或进行 **Auto-Configure** 后，调试器就必须与 Multi-ICE server 相连。
ARM 调试器的描述分为两个部分：
连接 AXD；
连接 ADW 和 ADU。
对于 Windows 和 UNIX 系统而言，其步骤都是一样的。图片是在 Windows 环境中获得，其指令还可用在 Solaris，HP-UX 和 Linux 系统中的 ADU 和 AXD 中。
注意：
根据默认设置，使用 Windows Explorer，文件打开对话框，隐藏文件的扩展名为.dll。除非此设置被修改，否则 Multi-ICE DLL 不会显示出来。
要在 Windows 95 或 Windows NT4.0 中，通过不升级桌面来显示.dll 文件，需要执行以下步

骤：

1 打开一个 Windows Explorer 窗口，选择 View→Options 选项；

2 选择 View 标签；

3 选择 Show all files 单选按钮；

4 点击 Options 对话框的 OK 按钮。

要在 Windows 95 或 Windows NT4.0 中，通过升级桌面来显示.dll 文件，需要执行以下步骤：

1 打开一个 Windows Explorer 窗口，选择 View→Folder Options 选项；

2 选择 View 标签；

3 在树状视图中，找到 Files and Folders→Hidden Files。选择该组中的 Show all files 单选按钮；

4 点击 Folder Options 对话框的 OK 按钮。

3.2.1 连接 AXD

在 Windows 或 UNIX 环境下使用此进程来激活 AXD 中的 Multi-ICE DLL。

1 选择 Options→Configure Target，如图 26 所示。

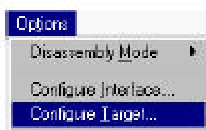


图 26 AXD 选项菜单

接着弹出 Choose Target 对话框，如图 27 所示。

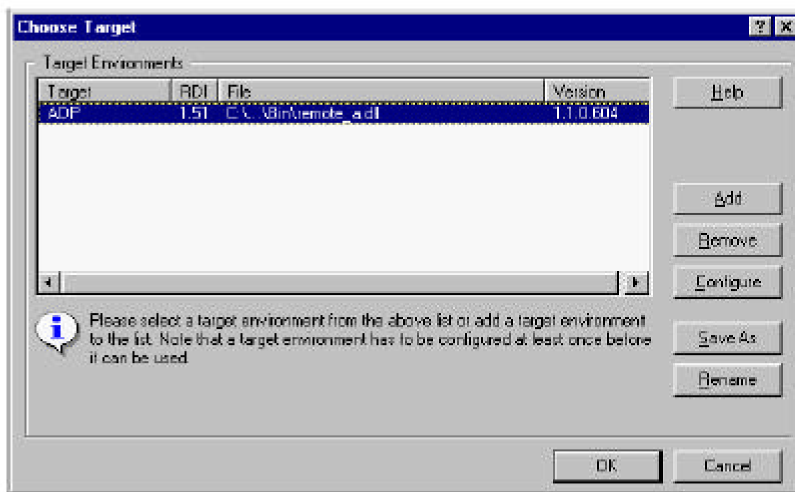


图 27 AXD 选择目标硬件对话框

2 如果在 Target Environment 列表中包含 Multi-ICE 的话，则选中它，并执行第 3 步。如果没有列出，则：

a 选择 Add。将显示一个 Windows 的 Open 对话框；

b 指到 Multi-ICE 的安装目录（比如，C:\Program Files\ARM\Multi-ICE）；

c 找到 Multi-ICE.dll 文件，选中它，然后点击 Open 对话框，如图 28 所示。

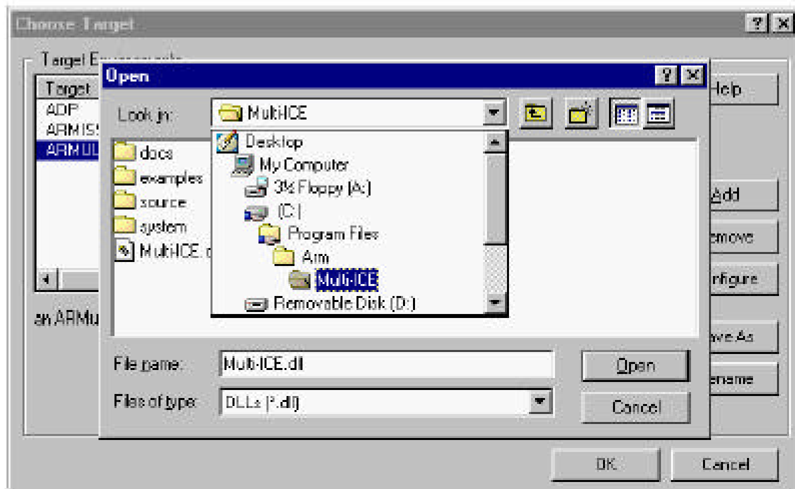


图 28　　　使用 AXD 选择 Multi-ICE DLL 文件

点击 Open，弹出对话框。

在 **Target Environment** 列表上将显示 Multi-ICE DLL 文件的路径。

3 选择 **Configure** 显示 Multi-ICE 的配置对话框；

4 开始配置 Multi-ICE DLL。

3.2.2 连接 ADW 和 ADU

使用此进程来激活 ADW 或 ADU 中的 Multi-ICE DLL：

1 启动 ADW 或 ADU。上个调试目标硬件使用的，或者是 ARMulator®（如果调试器刚刚安装），则将自动启动它们。如果弹出目标警告对话框，点击 No。ADW 或 ADU 将就会 ARMulator。

2 选择 **Options→Configure Debugger**，如图 29 所示。

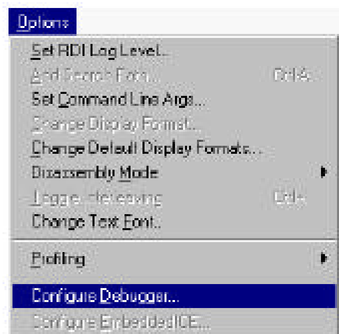


图 29　ADW 和 ADU 选项菜单

调试器将弹出一个调试器配置对话框，类似于图 30 所示。

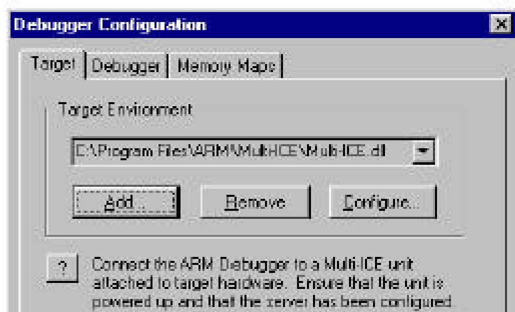


图 30　运行的 Multi-ICE 的 ADW 配置对话框

3 如果在如图 30 所示的 Target Environment 列表中已经列出了 Multi-ICE，则进行到第 4 步。如果没有列出，则：

a 选择 Add。将显示一个 Windows 的 Open 对话框；

b 指到 Multi-ICE 的安装目录（比如，C:\Program Files\ARM\Multi-ICE）；

c 找到 Multi-ICE.dll 文件，选中它，然后点击 Open 对话框，如图 31 所示。

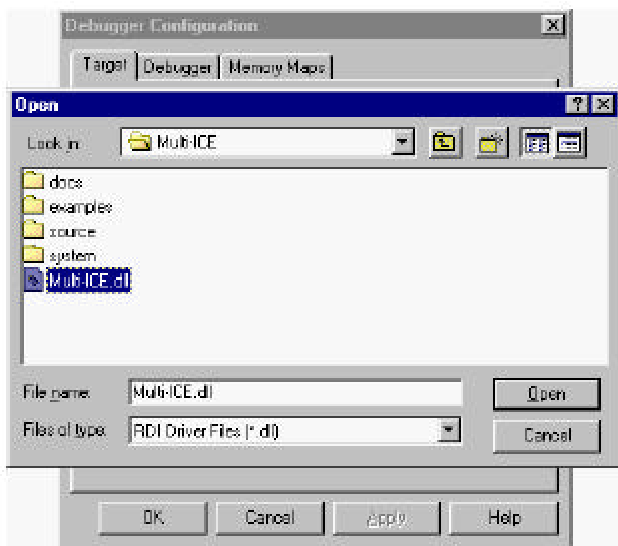


图 31　使用 ADW 选择 Multi-ICE

点击 Open，弹出对话框。

在 **Target Environment** 列表上将显示 Multi-ICE DLL 文件的路径。

4 选择 **Configure** 显示 Smart-ICE 的配置对话框；

5 开始配置 Multi-ICE DLL。

3.3 配置 Multi-ICE DLL

本节描述了 Smart-ICE 配置对话框的选项。包括以下内容：

* 连接配置；
* 处理器设置；
* 高级配置；
* 目标板相关配置；
* 跟踪配置；
* 关于 Smart-ICE;
* 查看通道配置。

最后一节介绍了如何保存配置设置：

* DLL 设置的保存。

3.3.1 连接配置

Multi-ICE server 的连接配置对话框如图 32 所示。

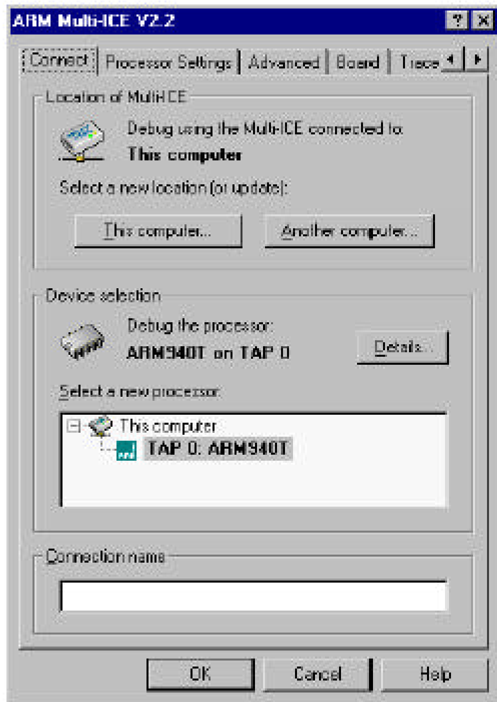


图 32　　Smart-ICE 连接配置对话框

如果没有成功配置 Smart-ICE，也会弹出其欢迎页面，如图 33 所示。

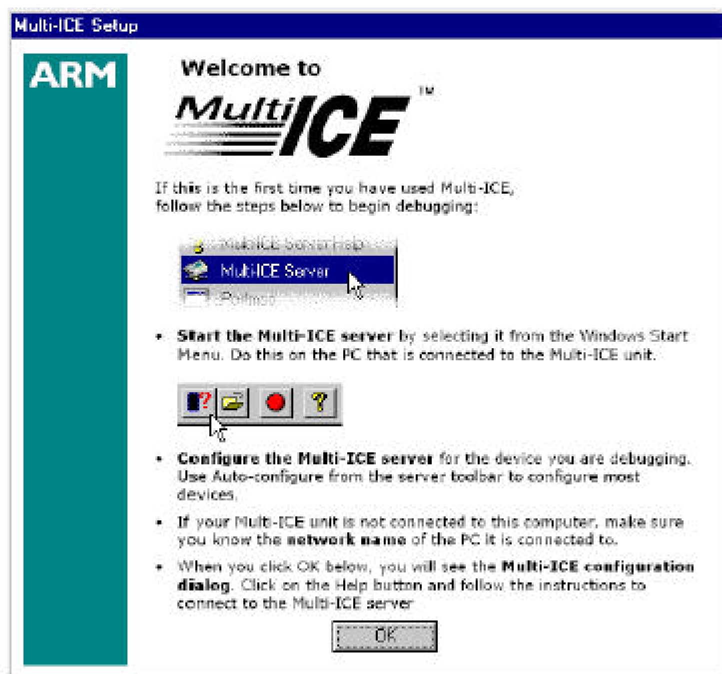


图 33　　Smart-ICE 的欢迎页面

此配置对话框包含以下组件：

Location of Smart-ICE

键入 Smart-ICE 相连的工作站名称，从而找到 Multi-ICE server 以及其仿真器，如果：

已经存在此连接，则将在设备选择区中显示该工作站，以及设备的详细信息；

还没有连接到 server 上，此区域和设备选择区为空白。

如果 Smart-ICE **仿真器连接到你正在使用的工作站，点击 This computer…按钮。如果没有** Multi-ICE server **在运行，则软件会询问是否启动一个** server。

如果 server **是在另外的工作站上运行，则点击 Another computer…。接着弹出 Select server location 对话框，你可以在 Network address 中输入** server **名称，或者在网络列表中选中它。**

Device selection

从设备树中选定你所期望的处理器设备。这些设备对应于在 Multi-ICE server **窗口的** TAP **配置区域中的显示内容，包括隶属于主设备的设备信息。**

如果你需要了解更详细的信息，点击 Details…。**（如图** 34 **所示）**

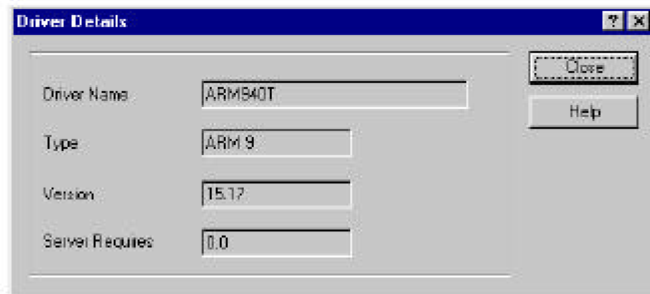


图 34　**驱动详细信息对话框**

此对话框又包含以下信息：

Driver Name　Smart-ICE **指定给设备的名称。与驱动信息文件** IRlength.arm **中使用的名称相同；**

Type　**处理器类型，比如，**ARM7，ARM9，Xscale **等；**

Version　**用来控制设备的软件驱动的版本号；**

Server Requires　**使用此驱动所需的** Multi-ICE server **的版本；**

Connection name　**其路径是可选的。如果你想自己给这个连接命名，就可使用此选项，这将帮助你确定使用此设备的工程师或测试程序。当在** server **上创建好连接后，此名称将显示在调试区中。**

要使你的设置生效，并连接到目标处理器上：

1 **点击** OK。**将返回到调试器配置对话框界面；**

2 **点击配置对话框的** OK **按钮，将连接到目标处理器上。如果连接成功，**server **窗口中的设备名称将变为红色，并且在** server **控制窗口中将显示连接信息。否则，就会弹出一条错误信息。**

远程 Multi-ICE servers

在连接配置标签上的 Another computer…**按钮将配置本地的** Smart-ICE **与另一台电脑上的** Multi-ICE server **的连接。显示的对话框内容取决于你的工作站上配置的网络软件：**

如果你有一个 Unix **工作站，或者一个** Windows **工作站，但没有安装局域网软件，请参看** *Workstations with no Network Neighborhood*；

如果你有一个 Windows **工作站，并且安装了网络软件，可以使用** Windows Network Neighborhood，**请参看** *Workstations with a Network Neighborhood*。

Workstations with no Network Neighborhood

在 UNIX 工作站和 Windows 工作站上，如果无法使用 Windows Computer Browser 服务，就将显示一个对话框，你可以在其中键入想要连接的机器名。

如果你使用的的是加载了 TCP/IP 协议的 Windows95，Windows98，或者 Windows Me，就必须在显示网络浏览对话框之前安装 Windows Computer Browser 服务。

Workstations with a Network Neighborhood

在可以使用 Windows Computer Browser 服务的 Windows 工作站上，允许你查看与 Multi-ICE server 连接的网络。在浏览器中只显示同时具备 Windows Computer Browser 服务和远程访问运行中的 Multi-ICE server 功能的工作站。

你可以通过两种方式来使用此对话框：

如果你知道想要连接的工作站，可以直接在 **Server name** 文本框中键入它的名称。你可以键入文本名（比如，PC2）或 IP 地址（以点状四元形式，比如 192.168.3.1）。然后点击 **OK** 完成；

你可以通过树状视图来搜索 servers。

要浏览网络，点击图标⊞，展开列表，直到你找到工作站的名称。这些工作站中有一个 Multi-ICE server 在运行，同时还启用了 **Allow Network Connections**（允许网络连接）设置。要将 Multi-ICE servers 列成一组，此组中的每个工作站都必须连接上。Smart-ICE 同时访问多个工作站，并且在对话框顶部显示相关信息。

如果没有访问到带有运行中的 server 的工作站，则这个组名称将不显示图标⊞。

当显示了一个机器，点击其名称旁的图标⊞，将展开在 server 上配置的处理器列表。

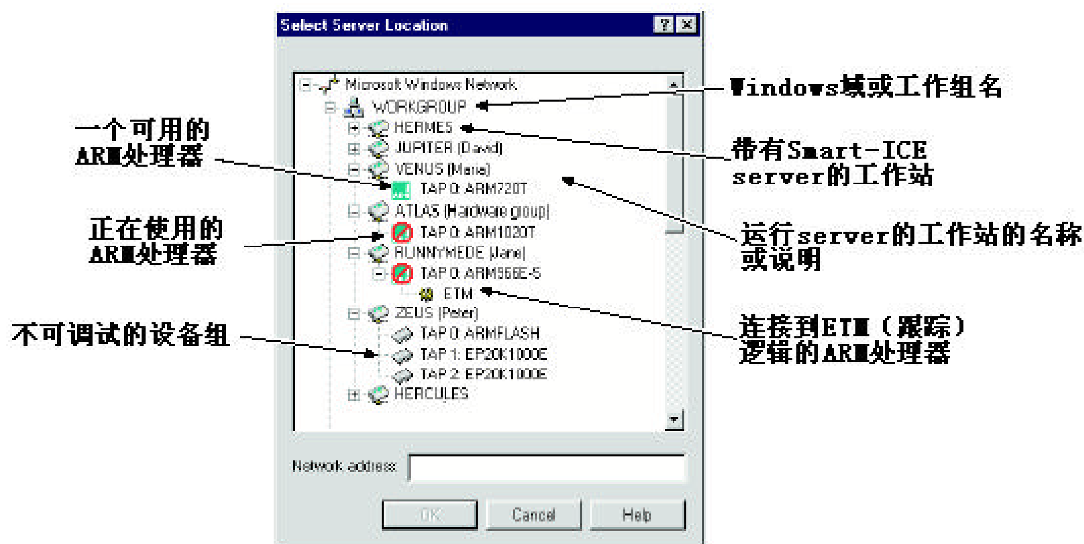


图 35　　Server 浏览对话框

设备名旁边的图标表示目标的连接状态：

图标上的红色圆圈表示一个带有激活连接的设备；

绿色的 ARM powered 图标表示 Smart-ICE 可调的 ARM 设备；

图标表示 Smart-ICE 不可调的 ARM 设备，比如 FPGA，DSP 芯片或 Flash 存储设备；

图标表示一个连接到同一 TAP 控制器上的扩展设备，比如一个

Embedded Trace Macrocell（ETM）。

如果你选定了一个要连接的设备，则在列表完成前点击 OK，显示内容不会有所增加。然而，当前的活动必须被完成，并且在进行时，一个文本信息将以 Stopping…开头。此过程大概继续几秒钟时间。

3.3.2 处理器设置

在你连接到目标硬件之前，处理器设置标签允许你改变处理器的特定设置。你定义的设置取决于要连接的处理器，并且在这个标签使用前你必须选定一个处理器（使用 Connect 标签）。你可以修改以下设置：

Cache clean code address

此项设置，是一个 128 位的存储器的基本地址。该存储器用来保存一个代码列，从而确保在 Data Cache（DCache）中的“脏的数据”被写入主存储器中。

清空 Dcache 非常重要，因为当处理器缓存功能启用后，要写入目标存储器的程序指令先写入到 Dcache 中，但要由指令缓存来读取。如果 Dcache 没有被清空，部分或全部指令在处理器执行它们之间就可能没有被写入主存储器中，结果该地址处的存储器的原先内容将被执行。

Smart-ICE 将按要求加载缓存清空代码。该区域必须是程序存储器，可以读写，并且不能用作其他用途。如果 Smart-ICE 没有向这个存储器中加载代码，你就会看到以下错误信息：

```
Could not clean D-Cache - memory may appear incoherent in writeback regions.
```

对于某些类型的处理器而言，其复位代码必须是可下载的，从而可以使 Smart-ICE 在遇到中断点或处理异常情况时可以复位处理器。对这些处理器而言，复位代码被写入缓存清空代码地址段中。

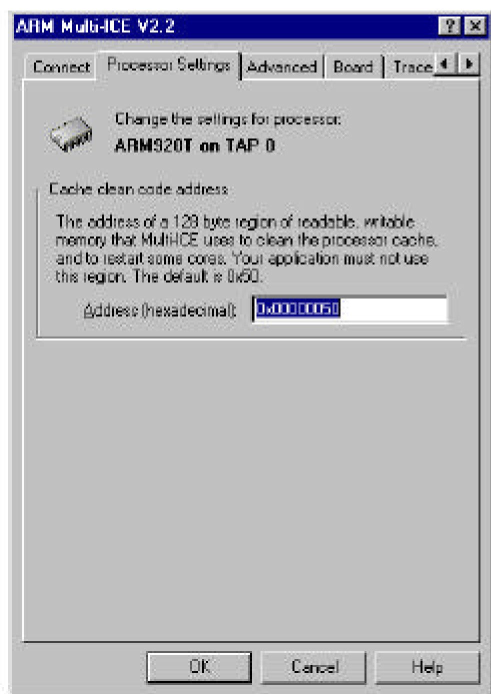


图 36 Multi-ICE 处理器设置标签的缓存设置项

Cache clean data address

此项用于 Xscale 的微结构处理器，它是 32KB 的存储器的基本地址。在清空处理器缓存时，Multi-ICE 将使用此项。
这个存储器：
必须是带缓存功能的；
必须同一个 32KB 的地址对齐（即，低位的 15 位地址必须为 0）；
大小必须为 32KB。

注意：此地址无需存在物理存储器。

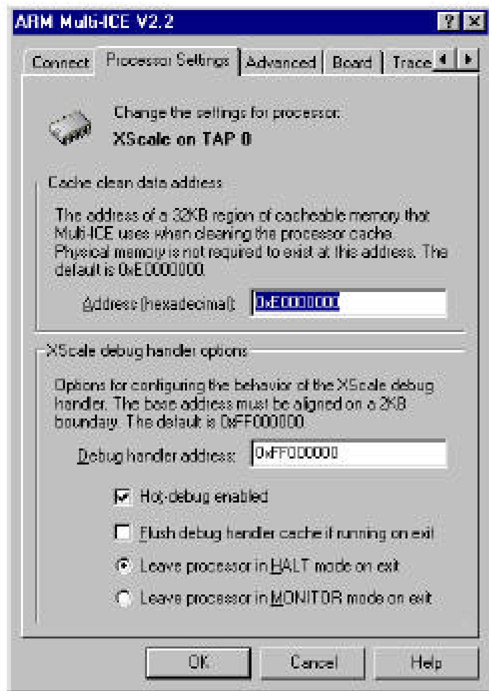


图 37　Multi-ICE 处理器设置标签的 Xscale 设置项

Xscale debug handler options
这些设置，如图 37 所示，用在 Xscale 微结构处理器中，配置调试句柄的相关动作。
包含下列相关配置：

Debug handler address
一个存储区域的基本地址，由调试句柄代码调用。此存储器：
必须与一个 2KB 地址对齐（最少的 11 位地址处为 0）；
大小必须为 2KB；
必须位于从地址 0 开始的 ARM 分支指令的范围内（大约±32MB）；
不能用作其他用途。
注意：此地址无需存在物理存储器。

如果你输入了不可用地址，将弹出一个消息框— The address you have entered is invalid.

Hot-debug enabled
一个启用或禁止 Smart-ICE 连接到一个已运行的 Xscale 微结构处理器上的触发器，它通过一

个调试句柄来实现：
如果你启用了热调试功能，ARM 提示目标硬件必须具备下列固件以适应此功能——如果你已经添加了固件支持，则要查看 Flush debug handler cache if running on exit 对话框，然后选择 Leave processor in Monitor mode on exit 按钮。

如果你禁用了热调试功能，Smart-ICE 在连接时通常会复位处理器。

Flush debug handler cache if running on exit
一个触发器，用以在退出调试器时确定缓存中的调试句柄是否清空并转存：
如果确定了该对话框设置，并且处理器在运行中，则调试句柄将从缓存中被清空并转存。调试器不能在稍后连接到句柄；
如果没有确定该对话框设置，则句柄不会被清空。调试器稍后将重新连接到句柄。
注意：
稍后改变了意外向量区的代码将不能正常执行。

Leave processor in Halt mode on exit，Leave processor in Monitor mode on exit
用以设置当退出调试器时，是否让处理器处于 Halt（暂停）或 Monitor（监控）模式：
如果你让处理器处于 Halt 模式，然后系统便会重新启动，调试句柄不会从缓存中被清空。处理器将暂停运行，直到调试器连接到句柄上；
如果你让处理器处于 Monitor 模式，然后系统会重新启动，调试句柄将从缓存中被清空，接着处理器将再次复位。

3.3.3 高级配置标签

高级配置标签包含允许你配置调试器的相关选项，用来匹配你的目标硬件的存储器字节顺序，是否包含缓存信息以及调试器软件接口方式等。如图 38 所示。

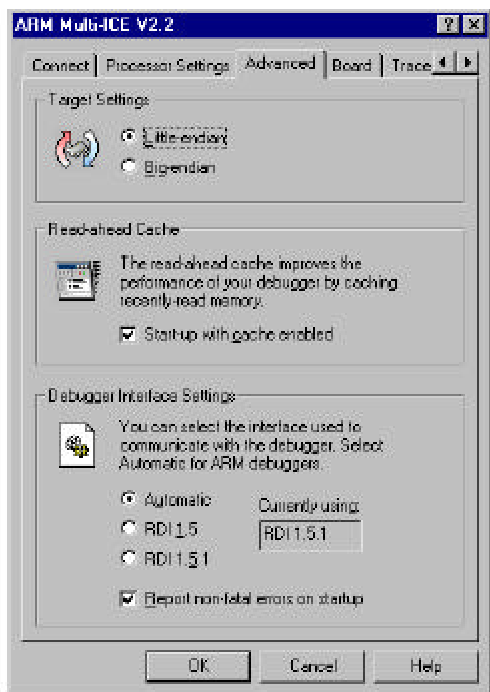


图 38　Smart-ICE 高级设置标签

对话框包含以下项目：

Target Settings

你可以指定是否让目标按 Little-endian（小端模式）或 Big-endian（大端模式）顺序排列，只需在目标配置对话框中按下 Target Settings **单选按钮即可。**

当你使用稍旧版本的软件时，这些按钮都不可用，比如 SDT2.51 **或** ADU**。你必须：**

1 **在你的调试器的 Options 菜单中选择 Configure Debugger（配置调试器）命令；**

2 **点击 Debugger 标签；**

3 **选择一个 Endian 选项按钮。**

Read-ahead Cache

当处理器停止运行时，此复选框允许你启用或禁用主机上的缓存目标存储器。默认设定值为 internal_cache_enabled**。你可以将调试器的中间变量** internal_cache_flush **设为** 1**，从而在某一时间开始清空缓存。还可以将** internal_cache_enabled **设为** 0**，暂时关闭缓存。**

快速读取缓存功能提高了存储器的读取能力，它可以读取比调试器要求的更多的数据，并根据需要将剩余数据放入缓存中。这将极大地提高某些操作的速度，比如当你需要执行一些数据（带有大量显示在调试器窗口中的串变量）。

初始化设置，DLL **并不执行快速读取。当一个存储区域成功执行了首个读取操作时，**DLL **将"意识"到可以很安全地访问此区域。接着它将此区域写入缓存，直到调试器复位。**

所有的 ADS **调试器将在程序间保存快速读取设置，但** SDT2.51 **的** ADW **版本不支持。**

注意：

默认设置是启用快速读取功能。当遇到以下情况时应将其关闭：

你尝试调试一个带有按需启用的存储器的系统；

使用调试器读取硬件寄存器的内容，而且你不想无意间读取附近的寄存器内容。

Debugger Interface Settings

Multi-ICE **支持遵循** RDI1.5.1 **的调试器，并且允许你连接到其他尊许此协议的调试器上。这个选项将显示正在使用的** RDI **的版本号。**

你可以通过该对话框选择需要使用的 RDI **版本。**

Automatic	**选择最适合调试器的模式，使用自动的版本签署协定。此项是默认设置；**
RDI 1.5	**要求** Smart-ICE **使用** RDI1.5**；**
RDI 1.5.1	**要求** Smart-ICE **使用** RDI1.5.1**；**

如果你使用的是 ARM **调试器，则建议你使用 Automatic 设置，因为它适用于所有的** ARM **调试器。**

注意：

AXD **只支持** RDI1.5.1**，且** Automatic **版本将自动选定** RDI1.5.1**。如果你要求** AXD **通过** RDI1.5 **连接到** Smart-ICE**，则** AXD **的** Stop **按钮不能使用。**

Report non-fatal errors on startup

此项设置与调试器首次连接到目标 Smart-ICE **时所生成的错误相关。如果你复选了相关选项，则严重错误和非严重错误都会被探测出来。这是默认的设置，并且将通知你所有由** Smart-ICE **探测到的问题。**

有些调试器认为在配置过程中探测到的错误为严重错误，如此将使你无法连接到目标硬件

上。如果发生此情况，你必须取消复选选项，从而使 Multi-ICE 只报告严重的错误信息。

3.3.4 目标板相关配置标签

目标板配置标签中有一个目标板的列表。当你在其中选定一个目标板时，调试器将显示外围设备的存储映象寄存器和板子上的微控制器。

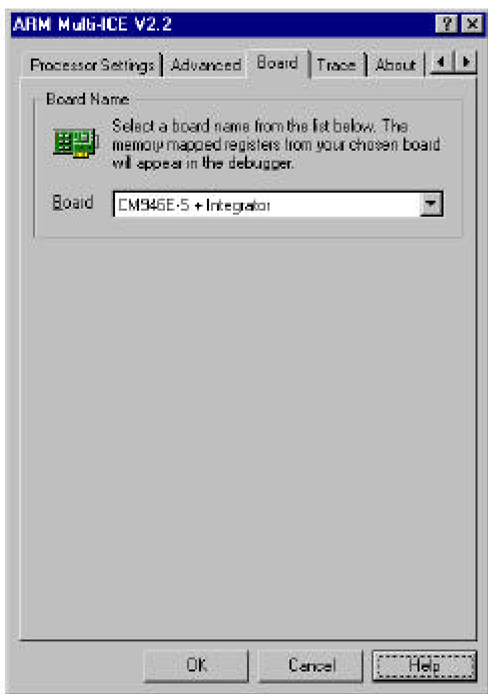


图 39　　目标板标签

注意：此标签只有在你使用了最新的支持此功能（如 ADS1.2 或稍后版本）的工具集后才会显示出来，但是你不能使用 RealMonitor。

如果你要使用 RealMonitor，则要设置支持 RealMonitor 的目标板方可。

3.3.5 跟踪配置标签

跟踪配置标签，如图 40 所示，必须在你的工作站上安装了 ARM *Trace Debug Tools*（TDT）才可显示。如果没有安装 TDT，则不能显示跟踪标签。

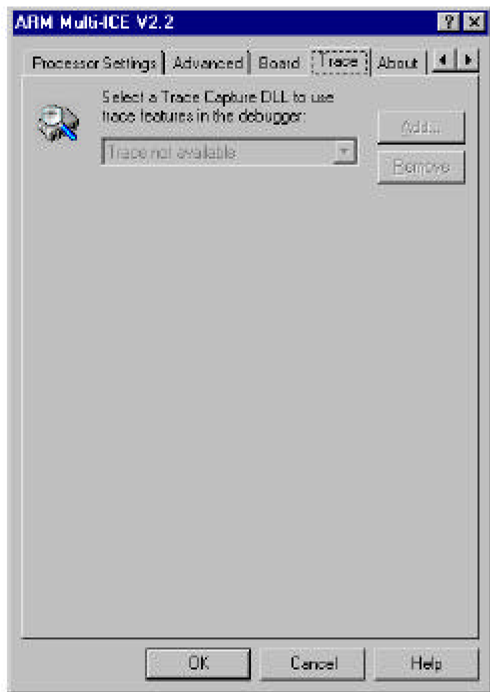


图 40　　　　跟踪配置标签

跟踪标签允许你配置跟踪组件，用来捕获跟踪信息。

3.3.6 关于 Multi-ICE 标签

你所使用的 Multi-ICE 的版本信息将显示在如图 41 的关于 Multi-ICE 标签中。它显示了 Multi-ICE DLL 的完整版本信息。

如果已安装并配置了 TDT，则用来控制捕获跟踪信息的 DLL 的版本号也将显示出来。如 multitrace.dll。

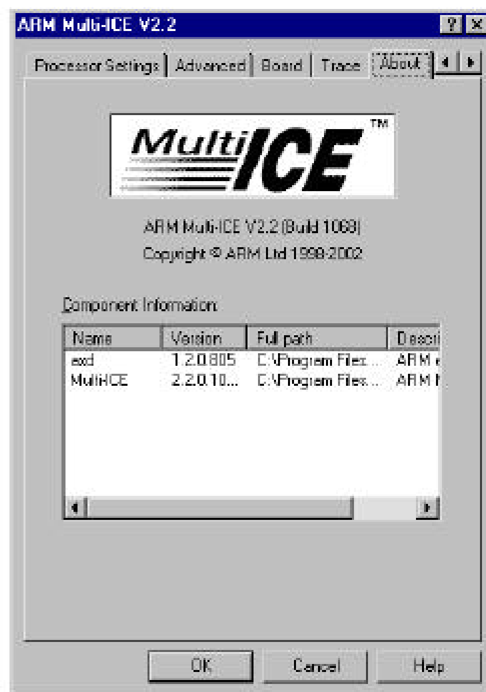


图 41　　关于 Multi-ICE 标签

3.3.7 查看通道配置标签

通道查看器允许信息从 DCC 中传送，并由调试器的外部程序执行。这是由 ARM 调试器提供的查看器，称为 ThumbCV。它负责通过 DCC 传送过来的语句的解释，并将它们显示在一个窗口中。它还有一个文本输入接口，允许你向目标板传送参数。

在 AXD 中使用通道查看器

Multi-ICE 配置对话框 **Channel Viewers** 标签只在使用 ADW 时才可用。

在 ADW 中使用通道查看器

通道查看器可以使主机以多种复杂方式监控目标硬件，或者对外部传感器进行仿真。

通道查看器控制着选定的通道查看器 DLL 的启用或禁用功能。如图 42 所示。

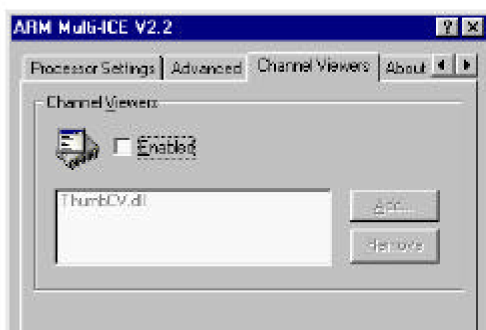


图 42　通道查看器控制

注意：

你不能同时使用通道查看器和 DCCsemihosting（将 semihosting_enabled 设为 2），因为它们使用的是同一 DCC。因此，如果你启用了 DCCsemihosting，再尝试添加或使用一个通道查看器时就无法完成操作；

你不能在 ARM10（Rev0）处理器或 Xscale 微结构的处理器上使用 DCC 通道查看器。

一共有三种控制方式。**Enabled** 复选框可以启用通道查看器功能（但同时也禁用了 DCC 的其他用途）。当选中了 **Enabled**，两个按键 Add 和 Remove 允许你对 Multi-ICE 上的通道查看器的 DLL 列表进行操作：

Add　　　　添加一个通道查看器的 DLL；

Remove　　　删除选中的通道查看器的 DLL。

添加查看器将使通道查看器进行初始化，比如系统提供的 ThumbCV 查看器，将在屏幕上创建一个新的窗口。

Multi-ICE 中的通道查看器的缓冲

Multi-ICE DLL 有一个内部的 1024 字的 DCC 缓冲器，用来从调试器到目标硬件间的数据传送。要传送到目标硬件的数据将暂时存储在这个缓冲器中，直到目标程序调用它。

3.3.8 DLL 设置的持续性

DLL 设置在运行间的持续性，以及它们如何保存，取决于你使用的调试器：

ADW（SDT2.51）

将大部分的 DLL 的运行间设置保存在寄存器中。然而，从 Multi-ICE1.3 开始对运行间设置不作保存，比如带缓存功能的快速读取复选框的状态；

ADW（ADS1.0.1）,ADU（ADS1.0.1）

将所有的 DLL 运行间设置都保存在用户档案文件中。比如，在 Windows NT 环境中，其文件名为：

c:\winnt\profiles*username*\adwtoolconf-*default*.cnf

如果你使用-session *name*参数来启动 ADS1.0.1 ADW，则各种操作的设置将保存在不同的文件中。这里 *default* 被换为 *name*。

AXD（ADS1.0.1）

将所有的 DLL 运行间设置保存在寄存器中。

AXD（ADS1.1 或稍后版本）

将所有的 DLL 运行间设置保存到寄存器中，或者在你选择 **File→Save Session** 时指定的文件中。

3.4 配置和调试多处理器

此节描述了使用 ADS v1.1（或稍后版本），以及 Multi-ICE v2.1（或稍后版本）的 AXD 调试器来创建一个多处理器系统。共分为两部分：

已命名的 AXD 目标硬件的配置方案；

进程文件的配置方案。

根据你的工作方式选择合适的模式。多目标硬件模式是最简单的模式，但进程文件模式允许更为自动的创建。

注意：

要在一个连接到 Smart-ICE 上的多处理器系统上使用 AXD，你必须在每个想调试的处理器上运行一个 AXD 的例程。每个 AXD 例程只能在一个处理器上运行。

3.4.1 使用已命名的 AXD 目标配置方案

AXD 允许你使用一个或多个已命名的目标配置方案。这非常有助于创建多处理器系统，因为你可以给每个处理器创建一个目标，并且在它们之间轻松地切换。

具体步骤为：

1 运行 AXD；

2 选择 **Options→Configure Target…**，将弹出目标配置对话框；

3 按下 **Add** 按钮，将 Smart-ICE 添加到目标列表中；

4 拷贝 Smart-ICE 的目标配置方案，以便在目标板上为每个处理器创建一个目标配置方案，如下：

a 在列表中选中 Multi-ICE；

b 点击 **Save As**，将弹出目标配置的保存对话框；

c 键入新的目标配置名；

d 点击 **OK**。

例如，要建立一个三处理器系统时，你可以创建另两个 Multi-ICE 的拷贝 Multi-ICE_TAP1 和 Multi-ICE_TAP2，然后点击 **Rename** 将原来的方案命名为 Multi-ICE_TAP0。

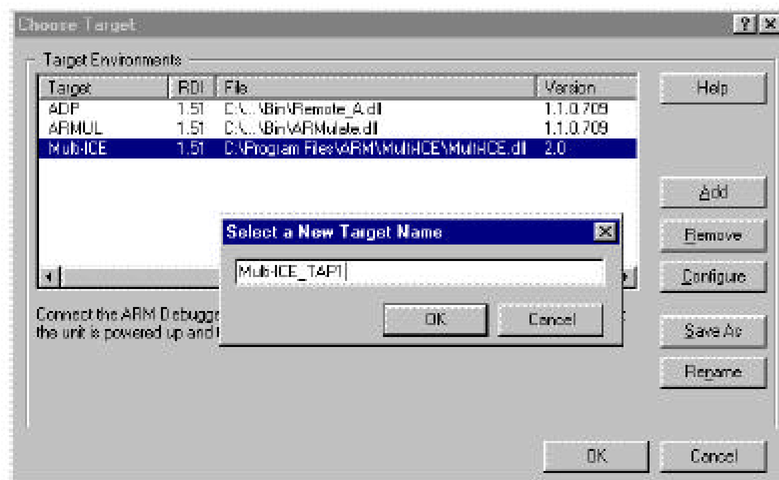


图 43　　保存一个已命名的目标配置方案

5 在 Target Environments（目标环境）列表中选择其名称，可以分别配置这些目标，然后点击 Configure，开始执行；

6 当你完成了每个目标的配置后，点击 AXD 目标配置对话框中的 OK 键。AXD 将保存所有目标的配置，并连接到你所选定的目标硬件上。

现在你可以在系统中轻松地切换处理器，只要在目标配置对话框中选择不同的目标即可。

配置 AXD 使其启动时选择一个目标

在默认配置中，当你启动 AXD 时它将自动选定上次所使用的目标硬件。你可以改变此设置，并使启动时将弹出目标配置的对话框，而不直接连接到默认目标上。要完成这些工作：

1 确认 Multi-ICE server 已经正确地配置好；

2 运行 AXD；

3 选择 Options→Configure Interface…，弹出接口配置对话框；

4 禁用 Session File 标签上的 Reselect Target 选项；

5 退出 AXD。

从 CodeWarrior 中启动 AXD

由 ADS 提供的 CodeWarrior IDE 允许你创建工程，并自动启动 AXD 来运行它。如果是多处理器系统，AXD 将尝试从最近使用的处理器上运行。

3.4.2 使用对话文件进行配置

如果你要求一个更为自动化的方式来运行多个调试器的话，可以在 AXD 中使用对话特性。一个 AXD 对话文件包含了调试器设置，比如一个可执行的映象文件名和窗口位置，甚至是当前的目标配置方案。AXD 的命令行变量允许你在启动时选择一个特定的对话文件。通过这个方式，可以实现自动地将 AXD 与一个指定处理器连接起来，只需改变你使用地对话文件即可。

但此方法不如在 *Configuration using named AXD target configurations*（使用已命名的 AXD 目标配置方案）一节中描述的多目标进程方式灵活，特别是当你需要频繁更改要连接的处理器时。然而，当你在启动 AXD 时指定一个处理器可以实现更好的自动化。

要配置首个 AXD 对话文件：

1 确认 Multi-ICE server 已经正确地配置好；

2 运行 AXD；

3 将 Multi-ICE 设置为目标配置方案。建议你点击 Remove 删除其他的配置方案，只保留 Multi-ICE，以避免进行深入修改时造成混淆；

4 对连接到目标板处理器上的 Smart-ICE 进行配置；

5 选择 Options→Configure Interface…；

6 启用 Session File（对话文件）标签里的 Reselect target（重选目标）功能。如果你没有启用此功能，则当你将这些对话文件重新加载到 AXD 中时，它们将不会连接到任何目标上；

7 点击 OK。

改变对话设定

现在你必须为系统中的每个处理器创建一个对话文件。具体步骤如下：

1 选择 Options→Configure Target…；

2 在目标配置列表中选中 Multi-ICE;

3 为那个处理器配置 Multi-ICE；

4 点击 Multi-ICE 配置对话框中的 OK 键，使配置生效；

5 点击配置目标对话框中的 OK 键，以连接到处理器上；

6 如果你想让 AXD 在连接到此处理器上时加载一个指定的可执行映象文件，则：

  a 点击 File→Load Image，以加载该文件；

  b 选择 Options→Configure Interface…；

  c 启用 Session File 标签中的 Reload Images（重新加载映象）功能；

  d 点击 OK。

7 如果你想在 AXD 启动时运行一个配置描述文件，比如要将处理器设在一个指定状态时：

a 选择 Options→Configure Interface…；

b 在 Session File 标签中选择 Run Configuration Script（运行配置描述文件）选项；

c 使用 Browse…按钮来定位配置描述文件的路径，或者直接在文本框中输入文件名；

d 点击 OK。

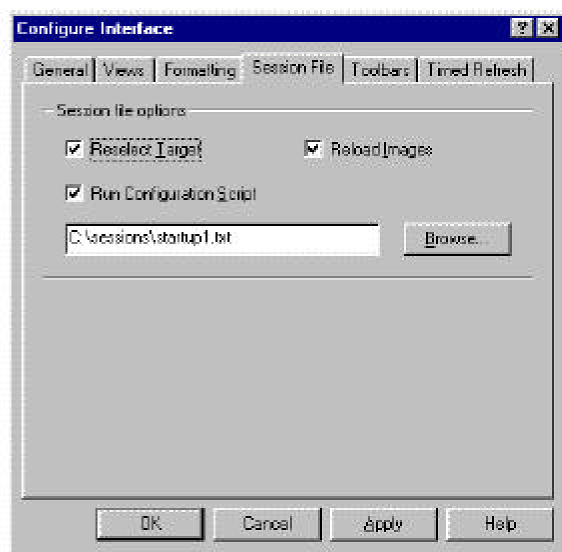


图 44 配置 AXD 以运行一个配置描述文件

8 选择 File→Save Session…以保存对话文件。你必须使用.ses 作为文件扩展名。建议你使用此配置方案连接的处理器名称为对话文件的名称。

并且你最好将对话文件放在一个方便访问的目录下。

现在你可以通过在 AXD 的命令行中指定一个恰当的对话文件，从而选定一个要连接的处理器。例如：

```
axd -session C:\sessions\tap1.ses
```

如图 45 所示，下面是三个 AXD 连接到一个 Multi-ICE server 和一个多处理器目标板的例子：

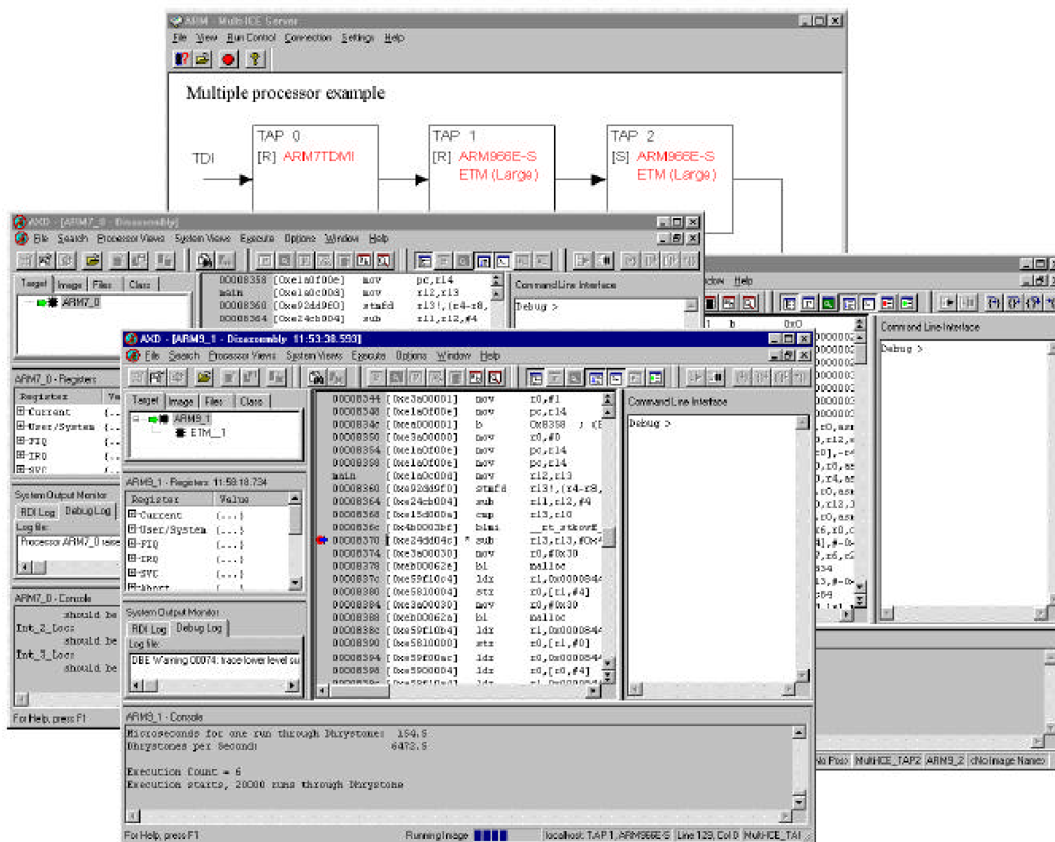


图 45　在一个多处理器目标板上配置三个 AXD 和 Multi-ICE server

将几个 AXD 对话文件命令放到描述文件中，从而可以自动启动所要求数量的 AXD 实例，以及一个连接到其他处理器的处理器。具体例子如下。Unix 描述文件命令 source ads.cshrc 将创建 ADS 可执行文件的路径和环境。这些文件的 Windows 版本位于 Multi-ICE 的安装目录中的实例目录下。

注意：

Windows 环境中使用了 pause 命令。因为多个 AXD 的拷贝不能同时加载到 Windows 中。Pause 命令使程序等待你输入一个按键指令，每个已加载和初始化的 AXD 拷贝完成后都必须这样做。

例 1　Windows 中的批处理文件

```
start axd -session C:\sessions\tap0.ses
pause
start axd -session C:\sessions\tap1.ses
pause
start axd -session C:\sessions\tap2.ses
```

例2　Unix 的命令描述文件

```
source ads.cshrc
axd -session sessions/tap0.ses &
axd -session sessions/tap1.ses &
axd -session sessions/tap2.ses &
```

你还可以在桌面上为每个处理器创建一个快捷方式。要在 Windows 中完成这些操作，步骤如下：

1 在桌面上点击鼠标右键，打开多级菜单；
2 选择 **New→Shortcut**（创建→快捷方式）；
3 在创建快捷方式对话框中，点击 **Browse**（浏览）；
4 使用浏览对话框来定位 AXD.exe 文件。如果你在默认位置上安装了 AXD，则其位于 `C:\Program Files\ARM\ADSv1_1\Bin\axd.exe`目录下；
5 点击 **Open**；
6 按下 End 键将光标移至行末，键入一个空格，接着输入：

```
-session "C:\sessions\tap0.ses"
```

则此行变为：

```
"C:\Program Files\ARM\ADSv1_1\Bin\axd.exe" -session "C:\sessions\tap0.ses"
```

7 点击 **Next>**；
8 给快捷方式取名，比如 AXD 或者 Tap0；
9 点击 **Finish**（完成）。

双击此快捷方式将启动 AXD，并且自动连接到你保存在对话文件中的相关设置的处理器上。你还可以在 Unix 桌面上进行类似操作。
注意：如果你想改变对话文件中的设置，必须对修改进行保存。AXD 不会自动将对话文件更新。

在 CodeWarrior 中进行对话

当你已经创建了对话，还可以对 CodeWarrior 进行配置，从而在点击运行时可以自动选定一个对话。如果每个处理器都有一个不同的 CodeWarrior 工程的话，这样做将最有效果。否则，你每次要使用另一个处理器时都必须对工程设置进行修改。
通过一个特定对话，对加载到 AXD 中的工程进行配置，步骤如下：

1 在 CodeWarrior IDE 中，调出工程设置对话框，在其树状控制设置的调试器部分中选择 ARM Runner；
2 选择 **Choose Debugger**（选择调试器）标签；
3 选择 **AXD**；
4 在 **Equivalent Command Line**（等效命令行）控制面板中，输入-session 参数。

例如：`axd -session C:\sessions\tap0.ses -exec &1`

注意：-session 参数和对话文件名必须放在一起，并且必须在-debug 或-exec 参数前。

5 点击 **Save**，关闭对话框。

警告：如果你稍后又改变了此控制面板中的其他设定，则必须再次输入-session 参数。

3.5 调试器内部变量

调试器内部变量经常出现在相关的调试器用户手册中。本节介绍了其他的调试器内部变量，即当你安装了 Multi-ICE 软件后可以使用那类变量。具体内部包括：

访问调试器内部变量；

处理器支持的内部变量；

内部变量的描述。

3.5.1 访问调试器内部变量

调试器内部变量是用来控制调试器行为或访问目标硬件的方式的值。有些变量控制前端调试器（比如，searchpath）。其他的则允许你控制 Smart-ICE 的操作，而不通过 Multi-ICE 配置对话框。

在 AXD 中，变量 vector_catch，vector_address，semihosting_enabled 和 semihosting_dcchandler_address 都可以通过处理器图标的右键菜单中 **Properties**（特性）菜单项来调用。

AXD 还有一个 **Debugger Internals**（调试器内部变量）窗口，你可以通过它来访问其他变量。

3.5.2 处理器支持的内部变量

可用的变量集部分取决于所选择的处理器。比如，与系统协同处理器相关的变量不能用在不含有系统协同处理器的处理器上。

下列表格列出了针对各个处理器组的可用变量。具体函数解释和变量所允许的值参看 *Internal variable descriptions*（内部变量的描述）。

标有Δ图标的变量不同用在 ADS v1.1（或稍后版本）中。AXD 和 Multi-ICE V2.1 都支持调试器所对应的目标硬件的建制，从而使这些变量显得并不必要。

标有Σ图标的变量与 AXD 的特征接口不匹配，也没有纳入处理器内部变量的列表中。

Variable name	ARM7[a]	ARM7T[b]	Samsung[c]	ARM7xxT[d]	ARM7EJ-S
cp_access_code_address	Yes	Yes	Yes	Yes	No
cp15_current_memory_area Δ	No	No	No	Yes[e]	No
icebreaker_lockedpoints	Yes	Yes	Yes	Yes	Yes
internal_cache_enabled	Yes	Yes	Yes	Yes	Yes
internal_cache_flush	Yes	Yes	Yes	Yes	Yes
ks32c_special_base_address	No	No	Yes	No	No
safe_non_vector_address	Yes	Yes	Yes	Yes	Yes
semihosting_dcchandler_address Σ[f]	No	Yes	Yes	Yes	Yes
semihosting_enabled =0 or =1 Σ	Yes	Yes	Yes	Yes	Yes
semihosting_enabled =2 Σ	No	Yes	Yes	Yes	Yes
sw_breakpoints_preferred	Yes	Yes	Yes	Yes	Yes
system_reset	Yes	Yes	Yes	Yes	Yes
top_of_memory	Yes	Yes	Yes	Yes	Yes
user_input_bit [1,2]	Yes	Yes	Yes	Yes	Yes
user_output_bit [1,2]	Yes	Yes	Yes	Yes	Yes
vector_address	No	No	No	Yes[g]	Yes[h]

表 5 ARM7 **系列支持的调试器变量**

a ARM7 **包括** ARM7DI，ARM7DMI **以及使用此类芯片的设备；**

b ARM7T **包括** ARM7TDI，ARM7TDMI，ARM7TDI-S，ARM7TDMI-S，**和使用此类芯片的设备；**

c Samsung **包括** KS32C50100 **和** S3C4510B**；**

d ARM7xxT **包括** ARM710T™，ARM720T™**和** ARM740T™**；**

e **只针对** ARM740T**；**

f **必须在** ARM **分支指令从** SWI **扇区计的范围之内（大约**±32MB)**。并且不能由从低位内存到高位内存分配的负分支确定；**

g **只针对** ARM720T**；**

h **没有系统协同处理器。**

Variable name	ARM9T[a]	ARM9xxT[b]	ARM9E-S	ARM9xxE-S[c]
cp_access_code_address	No	Yes	No	Yes
cp15_cache_selected Δ	No	Yes	No	Yes
cp15_current_memory_area Δ	No	Yes[d]	No	Yes[e]
icebreaker_lockedpoints	Yes	Yes	Yes	Yes
internal_cache_enabled	Yes	Yes	Yes	Yes
internal_cache_flush	Yes	Yes	Yes	Yes
safe_non_vector_address	Yes	Yes	Yes	Yes
semihosting_dcchandler_address Σ[f]	Yes	Yes	Yes	Yes
semihosting_enabled Σ	Yes	Yes	Yes	Yes
sw_breakpoints_preferred	Yes	Yes	Yes	Yes
system_reset	Yes	Yes	Yes	Yes
top_of_memory	Yes	Yes	Yes	Yes
user_input_bit [1,2]	Yes	Yes	Yes	Yes
user_output_bit [1,2]	Yes	Yes	Yes	Yes
vector_address	No	Yes[g]	Yes[h]	Yes

表 6　　ARM9 **系列支持的调试器变量**

a ARM9T **包括** ARM9TDMI™**和使用此类芯片的设备；**

b ARM9xxT **包括** ARM920T，ARM922T™，ARM925T™**和** ARM940T；

c **包括** ARM926EJ-S，ARM946E-S **和** ARM966E-S；

d **只针对** ARM940T；

e **只针对** ARM946E-S；

f **必须在** ARM **分支指令从** SWI **扇区计的范围之内（大约**±32MB)。**并且不能由从低位内存到高位内存分配的负分支确定；**

g **不包括** ARM940T（Rev 0）；

h **没有系统协同处理器。**

Variable name	ARM10[a]	XScale
cp15_cache_selected Δ	Yes	No
icebreaker_lockedpoints	No	No
internal_cache_enabled	Yes	Yes
internal_cache_flush	Yes	Yes
safe_non_vector_address	No	No
semihosting_dcchandler_address Σ[b]	Yes	No
semihosting_enabled Σ	Yes	Yes
sw_breakpoints_preferred	Yes	Yes
system_reset	Yes	Yes
top_of_memory	Yes	Yes
user_input_bit [1,2]	Yes	Yes
user_output_bit [1,2]	Yes	Yes
vector_address	Yes	Yes

表 7　ARM10 系列和 Xscale 体系调试器支持的变量

a 包括 ARM1020T 和 ARM10200T；

b 必须在 ARM 分支指令从 SWI 扇区计的范围之内（大约±32MB）。并且不能由从低位内存到高位内存分配的负分支确定。

3.5.3 内部变量的描述

以下是 Smart-ICE 允许你通过调试器使用的调试器内部变量列表：

`cp_access_code_address`

这将指定一个内存区域，至少 40 字节，由 Smart-ICE 在对协同处理器进行读或写操作时使用。Smart-ICE 保证此内存在使用后将加载其原始值。此内存区域必须是可读，可写和可执行的。

`cp15_cache_selected`

此项只能在 Harvard Architecture 的处理器上使用。包括 ARM9 和 ARM10 体系的芯片，但不包括 ARM7 体系类芯片。此外，在 XScale 处理器上也不能使用。

注意：

如果你使用 AXD，此变量将不可用。作为替代的，Smart-ICE 将目标协同处理器的寄存器名和读取信息解释给 AXD。AXD 在处理器寄存器视图中包含了被解释的寄存器，并允许你按要求对其值进行读取和修改。

它还对能完成读写操作的 CP15 寄存器进行解释。如下表所示：

Value	CPU type	Description
0	ARM9 or ARM10 (Harvard) cores	选择Data Cache（DCache）——数据缓存
1	ARM9 or ARM10 (Harvard) cores	选择Instruction Cache（ICache）——指令缓存
2	ARM946E-S, ARM966E-S	选择高双倍化指令存储器
3	ARM946E-S, ARM966E-S	选择高双倍化数据存储器

表 8　缓存选择类型值

`cp15_current_memory_area (0-7=Memory areas 0-7)`

此变量用来选择寄存器 6 中要访问的区域，并且这些区域位于支持多重保护区域的处理器上。如果必要的话，用 `cp15_cache_selected` **变量来选择一个数据或指令存储区域。**
注意：如果你使用 AXD，此变量将不可用。作为替代的，Smart-ICE 将目标协同处理器的寄存器名和读取信息解释给 AXD。AXD 在处理器寄存器视图中包含了被解释的寄存器，并允许你按要求对其值进行读取和修改。

`icebreaker_lockedpoints`

此变量控制用户访问 EmbeddedICE 逻辑查看点寄存器。它是一个 bitmask，bit 0 对应查看点单元 0，bit 1 对应查看点单元 1。如果处理器中嵌入了一个 IEU，则 bit 2 则对应 IEU 单元 0，bit 3 对应 IEU 单元 1，如此对应，直到 bit 31 对应 IEU 单元 29。
如果 bitmask 中的一个 bit 设为（1），则 Smart-ICE 将不使用其对应的查看点单元。如果设置为（0），Smart-ICE 则可以使用其对应的查看点单元。

`internal_cache_enabled (0=disabled, 1=enabled)`

此变量控制 Multi-ICE DLL 中的存储器缓存动作。当 AXD 启动时，它具有与 Smart-ICE 高级设置窗口中的 Cache Enabled（允许缓存）选项一样的值。用户可以修改这个值，进而禁用或启用缓存功能。

`internal_cache_flush`

此变量通常读为 0。将一个非零的值写入它将促使 Smart-ICE 的内部存储器缓存被清空。

`ks32c_special_base_address`

此变量包含 Samsung KS32C50100 或 Samsung S3C4510B 处理器中的特殊系统寄存器的地址。这些寄存器将被分配为一个多页存储器，并且不可能通过设备获知它们的所在。这个库中的单个寄存器由 Smart-ICE 的缓存操作代码调用。

`safe_non_vector_address`

此变量的默认值为 0x10000。它必须设在一个 64KB 内存区域的基址上，并且此区域不能与从 vector_address 起始的 64KB 内存区域相重叠。被调用的内存段必须设为 safe，因为在其中 Multi-ICE DLL 有可能引起一些访问本区域的操作，并且是无恶意的读取。内存读取不能

引起数据丢失，不能影响到 I/O 设备。Smart-ICE 不能向此区域内写入数据。

semihosting_dcchandler_address

当 semihosting_dcchandler_address 的值设为 2 时，semihosting_dcchandler_address 的值即 SWI 句柄的地址。

semihosting_enabled

此变量控制着 Multi-ICE DLL 中的 semihosting 功能。

sw_breakpoints_preferred

此变量控制选择中断点的算法。如果它的值非零，则选择中断点的算法将选择在任何可以使用软件中断点的地方（比如，它不会在 ROM 中设置软件中断点）使用软件中断点。

如果它的值为零，则选择使用硬件中断点，除非所要求的中断点数超过能提供的硬件中断点单元的数量。

system_reset

当进行读操作时，它通常为 0。如果写入一个非零值，则目标板将通过一个大约 250ms 的系统复位脉冲信号进行复位。

top_of_memory

此变量定义 C 库使用的堆栈空间的存储器的高位地址。SYS_HEAPINFO semihosting SWI 调用的结果将传送到目标板上。其默认值为 0x80000，意味着写入堆栈的第一个字的内容为 0x7FFFC。要了解 SYS_HEAPINFO 的用途，Smart-ICE 假设内存配置如下图所示：

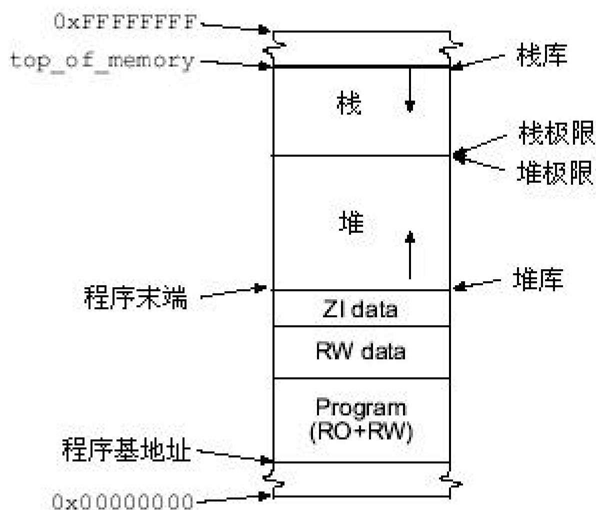


图 46　　top_of_memory 对应的单段的程序分布

注意：

如果应用程序是分段上载的，那么此应用程序必须包含一个用户定义的函数（__user_initial_stackheap），以定义栈和堆的极限。因此，如果应用程序没有调用 SYS_HEAPINFO，则目标将忽略 top_of_memory 的值；

top_of_memory 的值必须高于程序基址和程序大小的总和。如果没有设置正确，程序可能由于堆栈崩溃或程序盖写自身代码而无法运行；

当 top_of_memory 位于存储器的物理顶部时，无需其他要求。A C 或其他汇编程序可以在较高地址处使用内存。

user_input_bit1, user_input_bit2

这些变量显示了两个用户输入字节的状态。这些变量并非通信登记，因此它们将在 Debugger Internals（调试器内部变量）窗口显示时，芯片上次暂停时或者命令行命令执行时显示其状态。

user_output_bit1, user_output_bit2

这些变量允许你更改用户输出字节的状态。当它们被分配给这个连接并且 **Set by Driver** 选项被启用时，你只能修改其输出字节。这些变量可以通过 **Settings** 菜单中的 **User Output Bits** 项在 server 上进行设置。这些变量并非通信登记，因此它们将在 Debugger Internals（调试器内部变量）窗口显示时，或者命令行命令执行时显示其状态。

vector_address

此变量只适用于支持可移动扇区表的处理器，比如 ARM720T™和 ARM920T。当发现意外扇区表时，它将通知 Smart-ICE 该表所在的位置。其默认值是上次处理器停止运行所在的扇区地址。此地址由 CP15 寄存器 1 的 V 字节内容来确定。你可以将其设为 0 或者 0xFFFF0000。此地址上必须有可读内存。

3.6 Post-mortem 调试

此节介绍如何调试一个已经在运行但现在还没有连接到 Smart-ICE 上的系统的状态。比如，你可以找出为什么程序停止运行。

在你通过 Smart-ICE 来检查一个正在运行的目标硬件时，必须先配置 Smart-ICE 仿真器以及针对该目标硬件的 server。接着打开 Smart-ICE 仿真器的电源，然后配置 server，不影响到目标设备的状态。这要求 Smart-ICE 仿真器在连接到目标设备前就必须要加电。所以你必须使用外部电源给 Smart-ICE 供电，而不是从目标板取电。

3.6.1 通过电源插座给仿真器供电

Smart-ICE 提供的硬件有：

与 JTAG 接头相邻的电源输入插座；

电源调节和转换电路，允许你插入或拔掉 JTAG 线，而不影响到目标设备。

电源插座允许插入 2.1mm 的中央为正极的插头。你可以使用输出电压为 5V 的 DC 的电源，以及至少 300mA 的供电电流。

注意：仿真器的 JTAG 电路使用的电压由 JTAG 接头上的 **VTref** 信号决定。所以 **Vtref** 电压应该是 CPU I/O 接口的电压，如果该引脚没有连接到目标设备上，你必须修改目标板或者 JTAG 线。大多数情况下，将 **Vtref（JTAG PIN 1）**连接到 **Vsupply(JTAG PIN 2)**上即可。

要连接到一个正在运行的目标设备上：

1 确认 Smart-ICE 没有由目标设备供电；

2 JTAG 输入线 **TDI**，**TMS**，**nSRST** 和 **nTRST** 必须带有上拉电阻，并且 **TCK** 必须带有一个下拉电阻，以便当适配器没有与目标设备连接时，这些线依然保持它们的静止状态。

3 将电源插座插入到仿真器中；

4 配置 Multi-ICE server。你必须手动配置 server 或者使用一个独立测试系统来自动配置

server。然后让 server 自行运行；
注意：不要使用待调试的目标设备上的自动配置功能。如果这样做将使处理器复位。
5 将 20 位的 JTAG 线插入到目标设备中；
6 启动调试器。调试器可以停止处理器的运行，并显示其停止状态。
要获得该问题的更详细信息（源代码），你必须将目标程序的字符表加载到调试器中。对于 AXD 而言，使用 **File→Load debug symbols…**。对于 ADW 或 ADU 而言，使用 **File→Load symbols only…**；
7 按下 **Go** 或 **Run** 按钮，然后拔掉 JTAG 接头，复位系统。接着退出调试器。

3.7 访问 CP15

Smart-ICE 支持协作处理器。ARM 处理器的描述包含一个系统控制的协作处理器的描述，CP15，因此你不必将它们解释给调试器。

3.8 Semihosting

Semihosting 允许目标 ARM 处理器向运行调试器的电脑发送 I/O 指令。这意味着目标设备在开发过程中不需要屏幕，键盘或者硬盘。这些指令由调用 C 库函数来完成，比如 printf()和 getenv()。Smart-ICE 使用的 Semihosting 功能分为下列两部分：
*启用 Semihosting；
*使用 Smart-ICE 时，添加一个应用程序的 SWI 句柄。

3.8.1 启用 Semihosting

当使用 Multi-ICE DLL 时，一个真实的 SWI 意外句柄或使用中断点仿真一个句柄都可以执行 semihosting。你可以使用下列调试器中间变量来修改 semihosting 机制：

`semihosting_enabled`

根据默认设置，此变量设为 1 时，将启用中断点的 semihosting 功能。你还可以将它设为下列值：

0	禁用 semihosting
1	启用 start-stop semihosting，使用基于中断点的 SWI 句柄仿真
2	启用 DCCsemihosting，使用一个可以通过 DCC 与主机通信的意外句柄

在 vector_catch 中的 S 字节不能用作改变 `semihosting_enabled` 值的选项。

`semihosting_vector`

此变量控制 Multi-ICE DLL 设定的中断点的位置，继而侦测一个 semihosting 的 SWI。根据默认将它设为 8，除非 vector_address 指定使用更高的地址。
在 ADW 和 ADU 中，你可以通过 View 菜单的 Debugger Internals 项来访问这两项。在 AXD 中，调试器的中间变量由指定的窗口访问。

Start-stop semihosting

Start-stop 或标准的 semihosting 包括在 SWI 扇区上设置一个中断点或者通过你自己的 SWI 句柄指定位置来实现，取决于 semihosting_vector 的值。
当触发一个中断点后，Smart-ICE 将其认定为一个 semihosting 请求：
按要求读取处理器寄存器和存储器，以对请求指令进行解码；

请求指令在主机上执行；
返回值置于寄存器 R0 中，如果有要求，对存储器内容进行修改；
pc 被修改，因此下一条指令将是紧跟 SWI 的那一条；
程序运行恢复。

注意：当系统带有实时中断驱动软件时，不要使用 Smart-ICE 的标准 Semihosting 功能。因为一旦执行 semihosting 操作，处理器就将停止运行，中断也会被错过。要调试这些系统应该使用 DCC semihosting 或 ARM RealMonitor 功能。
并且在 SWI 扇区上的中断点将使用可能用作其他用途的中断点源代码。

DCC semihosting
DCC semihosting 具备两个优于标准的基于中断点的 semihosting 的功能：
大多数情况下它的运行速度最快；
它不会引起目标处理器进入调试状态，并且中断将继续有效。
标准的 semihosting 是初始的 semihosting 模式，因为 DCC semihosting 主要是嵌入目标设备的模式。
因为 DCC semihosting 不会引起处理器停止运行，此方式更适合于实时系统。在使用两个或多个处理器的 JTAG 链上它将更有用，因为 DCC semihosting 不会干扰处理器的自动启动和停止。
你不能将 DCC 用作其他用途（例如，频段查看），但 DCC semihosting 可以完成。
DCC semihosting SWI 句柄安装在目标存储器中，其地址放在 semihosting_dcchandler_address 变量中。它的重要性在于：
必须在 ARM 分支指令从 SWI 扇区计的范围之内（大约±32MB)。并且不能由从低位内存到高位内存分配的负分支确定；
调试器将句柄写入的存储器并没有使用。

SWI 句柄大小不过 0.75KB，并且可以在下列任一情况下写入到存储器中：

将semihosting_enabled的值设为 2，则启用 DCC semihosting；

semihosting_dcchandler_address的值被改变，并且已经启用了 DCC semihosting。

semihosting_dcchandler_address的默认值为 0x70000。要改变句柄的位置，你必须：

1 将semihosting_enabled的值设为 0，禁用 semihosting 功能；

2 将 semihosting_dcchandler_address设为一个新值，从而改变句柄的地址；

3 将semihosting_enabled的值改为 2，再次启用 DCC semihosting 功能。

注意：在使用 Rev C 或较早的 AMBA 包装的处理器中，你不能使用 DCC semihosting （semihosting_enabled=2)。则使用semihosting_enabled=1 （stop/start semihosting）来替代。

3.8.2 使用 Smart-ICE 时，添加一个应用程序的 SWI 句柄
很多应用程序在使用 semihosting SWI 的同时还使用自己的 SWI 句柄。你必须遵照以下步骤

才能使应用程序的 SWI 句柄与 Smart-ICE 的 semihosting 机制相匹配：

1 在扇区表中安装应用程序的 SWI 句柄；

2 对标准的 semihosting 而言，修改semihosting_vector的值以使它指向当你的句柄无法识别 SWI，或将其认定为一个 semihosting SWI 时的位置；

3 对 DCC semihosting 而言，在 SWI 扇区中安装应用程序的 SWI 句柄。当应用程序无法识别一个 SWI 时，它将跳转到带有处理器状态信息的semihosting_dcchandler_address+12处，以此调用 SWI 句柄。DCC semihosting 句柄将处理请求指令，并返回一个调用代码。

例如，一个特定的 SWI 句柄可能探测到当它无法处理一个 SWI，或者跳转到了一个错误的句柄处。

```
                             ; r0 = 1 if SWI handled
    CMP r0, #1               ; Test if SWI has been handled.
    BNE NoSuchSWI            ; Call unknown SWI handler.
    LDMFD sp!, {r0}          ; Unstack SPSR...
    MSR spsr_cf, r0          ; ...and restore it.
    LDMFD sp!, {r0-r12, pc}^ ; Restore registers and return.
```

例 1　　基本的 SWI 句柄

你可以修改这些代码以配合 Smart-ICE start-stop semihosting 的使用，如例 2：

```
                             ; r0 = 1 if SWI handled
    CMP r0, #1               ; Test if SWI has been handled.
    LDMFD sp!, {r0}          ; Unstack SPSR...
    MSR spsr_cf, r0          ; ...and restore it.
    LDMFD sp!, {r0-r12, lr}  ; Restore registers.
    MOVEQS pc, lr            ; Return if SWI handled.
Semi_SWI
    MOVS pc, lr
```

例 2　　带有 Smart-ICE 连接的 SWI 句柄

你必须创建一个带有地址 Semi_SWI 的 semihosting_vector。位于此地址的指令实际上却从不被运行，因为 Multi-ICE DLL 在运行完 semihosted SWI 后将直接返回到应用程序中。使用通用的 SWI 返回指令可以确保在没有创建 semihosting 中断点的情况下，不会引起应用程序崩溃。

如果应用程序连接了 semihosted ARM C 库，则会使用 C 库的启动代码，你必须在应用程序安装自己的句柄前修改 semihosting_vector 的内容，主要是指在主程序代码中设置一个中断点。这是因为，如果在应用程序开始运行之前 semihosting_vector 被设置在了应用程序 SWI 句柄的 fall-through 段，那么在库初始化调用 semihosted SWI 时就会触发一个未知的查看点错误。在这一点处，SWI 扇区没有应用程序句柄写入，而且还可能仍然存在软件性中断点。于是就触发一个 Multi-ICE DLL 未知的中断点，因为 semihosting_vector 地址已经移到一个无法访问的地方。

注意：

如果你的应用程序不需要使用 semihosting，包括其启动代码，只需将 semihosting_enabled 设为 0 即可。

当移动一个曾经在 Smart-ICE 系统上由 Angel 调试监视器运行的应用程序时，必须要非常注意。在 Angel 调试监控系统中，以移动和修改 Angel 安装的 SWI 扇区内容，将应用程序的

SWI 句柄安装到 SWI 扇区中来创建 SWI 句柄。此方法不适用于 Multi-ICE DLL，因为没有指令从 SWI 扇区中移出，并且没有代码输入。因此，当把一个应用程序移到基于 Smart-ICE 的系统上时，你必须改变 Smart-ICE 安装应用程序和 semihosted SWI 句柄的方式。

3.9 查看点和中断点

ARM 调试器提供对 Smart-ICE 目标的中断和查看功能。以下部分为这些功能的具体介绍：

* 查看点；
* 中断点；
* 查看点，中断点和程序计数器；
* 扇区中断点和意外；
* 在 ROM 的 0x0 地址处进行扇区捕捉；
* 停止处理器的运行。

3.9.1 查看点

所有的 ARM 调试器查看点都是可修改数据的查看点。即，如果读取或写入的数据点的值与当前内存中的数据值一致，则它们不会被激活。

硬件查看点与软件查看点间的区别

硬件查看点是通过 EmbeddedICE 逻辑点探测写入到掩码地址的数据来实现的。这类查看点很有效率，因为程序只在相关数据被写入后才停止运行。但是它与一个 EmbeddedICE 逻辑点完全结合在一起。

注意：

如果查看一个结构或阵列，掩码将引入某些未查看部分的地址。在这种情况下，向这些不需要的地址中写入数据的操作将被调试器过滤掉。程序运行速度将稍稍减慢，因为当触发了不需要的中断点时，处理器将停止运行，然后再由调试器自动复位。

软件中断点则以不同的方式来使用 EmbeddedICE 逻辑点。在执行了每条指令后，数据地址将被逐个检验，以观察它们的值是否变化。如果某个值发生变化，程序就将停止运行。否则，程序重新开始运行。这类查看点将极大地缩减程序的运行内容。此外，它不能用在只能写入的内存区域，比如一些映象配置的存储器设备寄存器。

3.9.2 中断点

当你查看当前的中断点和查看点（在 ADW 命令窗口中使用不带自变量的查看或中断命令，或者查看 ADW，AXD 和 ADU 的 **Breakpoints** 或 **Watchpoints** 窗口），其输出内容将指定它们是否是硬件或软件中断点，抑或是查看点。

硬件中断点与软件中断点间的区别

硬件查看点是通过 EmbeddedICE 逻辑点探测一个恰当地址的指令来实现。它适用于任何情况，甚至在程序运行时对其进行调试修改，或者是 ROM 中的代码。但是，它与两个可用的 EmbeddedICE 逻辑点单元完全结合在一起。

对优于 ARM v5 的芯片而言，软件中断点是通过一个 EmbeddedICE 逻辑单元探测某个特定二进制模式的指令来实现。这个二进制模式已经存储在一个恰当位置，同时真正的指令则存储在主机的调试器内存中。使用单个 EmbeddedICE 逻辑点可以支持任意个软件中断点。

ARM v5 芯片有特殊的中断点指令，因此不需要额外的 EmbeddedICE 逻辑点。

自调试代码，ROM 中的代码，或从磁盘文件中分出的代码，都不能使用软件中断点来调试。如果你尝试将一个软件中断点放在 ROM 中，Smart-ICE 就会探测到存储器不可被写入，从而尝试使用硬件中断点。

3.9.3 查看点，中断点和程序计数器

当被查看的数据发生变化时，查看点就被触发。当发生这类情况时，程序计数器就会跳转到触发查看点的指令处。被查看的数据此时已经是新值，而不是以前的值。
当待中断的指令进行“执行”状态时，中断点被触发，但是在此指令运行前。因此，当中断点被触发，程序计数器并没有更新，并保持在被中断指令的地址处。
注意：
在一个 ARM CPU 的芯片中，程序计数器将指向当前运行指令外的两条指令处。（实际上，它是当前被设置为 Fetch 状态的指令地址）ARM 调试器将简化这一过程——将一个修改过的值传送给程序计数器,从而当它在调试器中显示的内容为正在执行或将要执行的指令的地址。

3.9.4 EmbeddedICE/RT 中断点

EmbeddedICE/RT 逻辑是 EmbeddedICE 逻辑的升级版本。它被纳入了从 Rev4 开始的 ARM7TDMI 处理器和从 Rev2 开始的 ARM9TDMI 处理器中，并支持实时调试。很多改进措施都是针对基于目标的监控器，比如 RealMonitor。Smart-ICE 可以在目标程序运行时使用 RT 扩展来设置中断点和查看点。当然必须是目标调试器也支持此功能才行。带有 ADS v1.1 或稍后版本的 AXD 都支持此功能。

3.9.5 扇区中断点和意外

Multi-ICE DLL 将按照收到请求指令的顺序设置中断点，包括由调试器内部变量 vector_catch 发出的请求指令。Multi-ICE DLL 使用尽可能高效的中断点源代码，但有时执行一个扇区捕捉中断点时需要运行一个软件中断点。
vector_catch 变量决定当某个条件达到表 9 中的值时，是否执行捕捉。ADW 的默认值是 %RUsPDAifE，AXD 的默认值是 %RUsPDif，大写字母表示条件被截取。

Vector	Description
R	Reset
U	Undefined instruction
S	Software interrupt (SWI)
P	Instruction prefetch abort
D	Data access abort
A[a]	Address exception
I	Interrupt request (IRQ)
F	Fast interrupt request (FIQ)
E[b]	Error

a. 不能由AXD使用
b. 不能由AXD或ADW使用

表 9　　中断点

在 ARM9TDMI 和 ARM10TDMI 系列设备中，以及 XScale 微结构处理器中，芯片中的额外硬件允许你在无须在扇区上设置通用中断点的情况下，进行扇区捕捉。
作为不同用法，vector_catch 中的 SWI 字符仍然保持小写，但调试器内部变量 semihosting_enabled 和 semihosting_vector 却能提供更好的控制。
注意：如果设置了 vector_catch 中的 S 字节，并且将 semihosting_enabled 设置为非零结果，则扇区捕捉功能将优先于 semihosting。

3.9.6 在 ROM 的 0x0 地址处进行扇区捕捉

在系统中，位于地址 0x0 的 ROM 内，你必须注意 vector_catch 的设置。

3.9.7 停止处理器的运行

有两种方式停止处理器的运行：
使用 DBGRQ，并等待 DBGACK。这是标准方式；
使用 nSRST，并插入 nTRST，在复位扇区中设置一个中断点，然后释放 nSRST。这是备用方式。
如果标准方式失败了，系统将提示是否使用备用方式。此操作将复位芯片和 TAP 控制器，并且让 TAP 自主运行。接着它将在地址 0 处运行一个硬件中断点，然后释放系统复位命令，并等待处理器触发中断点。
复位方式对芯片没有时钟时很有用，比如执行了一个 nWait，因为它允许调试器重新控制处理器。
注意：
备用方式比标准方式对硬件要求更高，因为它要复位一个多处理器系统中的所有处理器。

3.10 缓存数据

Smart-ICE 处理缓存数据的方式取决于被调试的处理器类型：
如果你调试的是 ARM7，ARM9 或 ARM10 芯片，则参看 ARM 微结构处理器上的缓存数据一节；
如果你调试的是 XScale 微结构处理器，则参看 XScale 微结构处理器上的缓存数据一节。

3.10.1 ARM 微结构处理器上的缓存数据

当调试 ARM7，ARM9 或 ARM10 芯片的缓存处理器时，Smart-ICE 将保留尽量多的缓存内容。在理想情况下，使用以下方式：
当处理器进入允许缓存功能的调试状态，Smart-ICE 将：
禁用 ICache 和 DCache 的行填充功能；
保存当前 DCache 的写入属性；
选择 DCache 的 write-through 属性。
如果数据是在调试状态下的可缓存存储器中读取的，Smart-ICE 不会将它读入缓存中，因为行填充功能已经被禁用。
如果数据是在调试状态下被写入存储器中，Smart-ICE 将执行以下操作：
1 使数据中的包含地址的 ICache 所属的所有缓存行都无效。而其他的缓存行则不受影响；
2 更新数据中的包含地址的 DCache 所属的所有缓存行的数据值。它将数据写入内存中。其他的缓存行不受影响。
注意：
如果数据中的地址没有处在优先于写入的缓存中时，则不能添加到缓存中，因为行填充功能

被禁用。

当处理器跳出调试状态，Smart-ICE 将执行以下操作：
启用 ICache 和 DCache 的行填充功能；
恢复保存的 DCache 的写入属性。
注意：
缓存预留的效率取决于实际使用的处理器。在某些情况下，处理器设计上的限制使得 Smart-ICE 不能达到理想的缓存预留效果。但是，它可以保证没有数据丢失，并保持缓存一致性。

锁定数据
如果你将位于锁定段的缓存行设为无效,则此行中的废数据将全部丢失。锁定效用依然存在。因为缓存行被设为无效，其他行无法触发此行，即时退出调试状态后，它也保持在闲置状态下。任何对此无效地址的访问都将使用没有锁定的缓存区域。

3.10.2 XScale 微结构处理器上的缓存数据
当调试一个 XScale 微结构的缓存处理器时，Smart-ICE 将执行以下操作：
清空整个 DCache 的内容。
如果数据是在调试状态下的可缓存存储器中读取的，Smart-ICE 会把它读入缓存中，因为行填充功能没有被禁用。
如果数据是在调试状态下写入到存储器中，Smart-ICE 将执行以下操作：
1 清空整个 ICache；
2 更新数据中的包含地址的 DCache 所属的所有缓存行的数据值。它将数据写入到内存中，并对写入进行仿真。其他的缓存行不受影响。

锁定数据
所有的 XScale 微结构处理器中，锁定段不会执行清空和使无效的操作。继而进入调试状态，锁定段中的缓存行没有被清空。如果你禁用了调试器的缓存功能，试图写入锁定缓存区的数据将可能丢失。

3.11 在 ROM 中调试程序
这一节介绍的是关于在 ROM 中使用 Smart-ICE 调试应用程序的内容，具体分为以下部分：
复位后开始调试；
在 ROM 的零地址处调试系统。

3.11.1 复位后开始调试
你可以使用 Multi-ICE DLL 调试在 ROM 中运行的系统。特别是当一个在 ROM 中存有应用程序的目标板加电时，该程序开始运行。因此，当调试器在主机上已经启动后，目标板上的处理器将停止运行。在这个阶段，该应用程序可以位于其运行周期的每个点，但取决于调试器何时启动。
这意味着你可以检查系统的状态，并在当前位置重新启动程序。在某些情况下，这样做已经足够。但是，很多情况下，最好通过目标板的电源开关来实现程序复位。有两种方式：
复位仿真；
执行实际复位。

当你调试从 ROM 运行的代码时，确认至少有一个可用的查看点单元以保证在 ROM 中的代码上设置中断点，因为你不能使用软件中断点。不使用标准 semihosting 或者扇区捕捉功能可以减少调试器占用的单元数量。要使一个处理器去掉硬件性扇区捕捉功能，你必须在调试器启动后尽快将调试器中间变量改为下面的值：

```
semihosting_enabled = 0
vector_catch = 0
```

接着在设置非 ROM 中断点或查看点之前创建 ROM 中断点。否则查看点单元将被全部占用，从而使 ROM 中断点失效——系统将弹出一条警告信息，告知你已经有足够个数的中断点存在。

在 ROM 中调试系统的另一个关键点在于，ROM 映象文件并不包含任何调试信息。当使用 Multi-ICE DLL 进行调试时，通过将主机上的一个文件的相关信息加载到调试器中，从而使字符或源代码信息可用。比如使用 **Load Symbols…**命令。

复位仿真

你可以在调试器中进行以下设置，以实现一个复位仿真：

将 pc 置为 0（复位扇区的地址）；

将 cpsr 置为%IF_SVC（禁用中断，并进入超级用户模式）。

这模拟的是 ARM 芯片在通电或复位时的状态，但它不允许复位时配置内存或对目标硬件的特征值（比如外围寄存器）进行初始化。如果必要的话，你可以在运行应用程序前对这些特征值进行修改。你还可以使用调试器的脚本功能使这一过程实现自动化，如下面的例子所示。名称 embed.axf 必须由目标板所运行的文件名替换。也许还要对 top_of_value 值进行修改，但取决于你的目标板的内存排布。

```
readsyms embed.axf
pc = 0x0
cpsr = %IFt_SVC
$vector_catch = 0
$semihosting_enabled = 0
$top_of_memory = 0x40000
```

例1　　在 ADW，ADU 的 ADS 中推荐使用的复位脚本

同样的用在 AXD 的脚本如例 2：

```
loadsymbols embed.axf
setpc 0
sreg cpsr 0xd3
spp vector_catch 0
spp semihosting_enabled 0
let $top_of_memory 0x40000
```

例 2　AXD 中推荐使用的复位脚本

执行实际复位

取决于复位电路设计，你有可能要进行一次目标板的实际复位操作。但是，当它完成后要非常注意，因为如果 EmbeddedICE 逻辑也复位了话，调试器就不能执行同步操作。目标板要求两种复位的方式：

通过电源开关，使目标板上所有程序复位；
通过目标板上的复位键，使除了 EmbeddedICE 逻辑之外的程序复位。
如果在复位扇区上要求使用一个硬件中断点（或者是复位代码的起始地址），则应该使用推荐的复位电路。当目标板复位后，它将被中止运行。
注意：如果因为硬件线路的原因而导致 Smart-ICE 不可用，则目标板要执行复位操作。在复位芯片前你应该删除所有的软件中断点和查看点。

使用 ARM 集成板的例子
ARM 集成板可以运行两种复位。复位开关可以执行所需的初始化复位，因此通过复位可以启用调试功能。但要求是要设置硬件中断点，并按下 Reset 键。

3.11.2 在 ROM 的零地址处调试系统
如果调试处理器时没有启动扇区捕捉硬件,并且是在 ROM 而非 RAM 的零地址处进行调试，你就必须将 vector_catch 设为 0。这样做可以避免 Smart-ICE 在扇区表中创建软件中断点。

3.12 直接访问 EmbeddedICE 逻辑
要操控 EmbeddedICE 逻辑寄存器，你可以使用显示与设置协作处理器寄存器的命令，并将协作处理器号码指定为 0。
注意：
XScale 微结构处理器没有包含一个 EmbeddedICE 逻辑段，并且这些处理器的协作处理器 0 是真实的硬件协作处理器；
EmbeddedICE 逻辑协作处理器不能由 ARM10 系列芯片模拟，因为其 EmbeddedICE 逻辑不同。
下列章节介绍了如何通过调试器访问和使用 EmbeddedICE 逻辑寄存器：
从 AXD 中读取 EmbeddedICE 逻辑寄存器；
从 ADW 中读取 EmbeddedICE 逻辑寄存器；
使用 EmbeddedICE 逻辑值；
ICE 扩展单元的支持。

3.12.1 从 AXD 中读取 EmbeddedICE 逻辑寄存器
要使用 AXD GUI 来访问协作处理器 0 的寄存器，选择 **Processor Views→Registers**。将弹出寄存器窗口，在其中你可以选择协作处理器 0 的寄存器。
注意：如果你使用的是 ADS v1.0.1，则窗口中的 EmbeddedICE 逻辑寄存器并没有单个命名，而是以 CoProc 0 组形式出现。

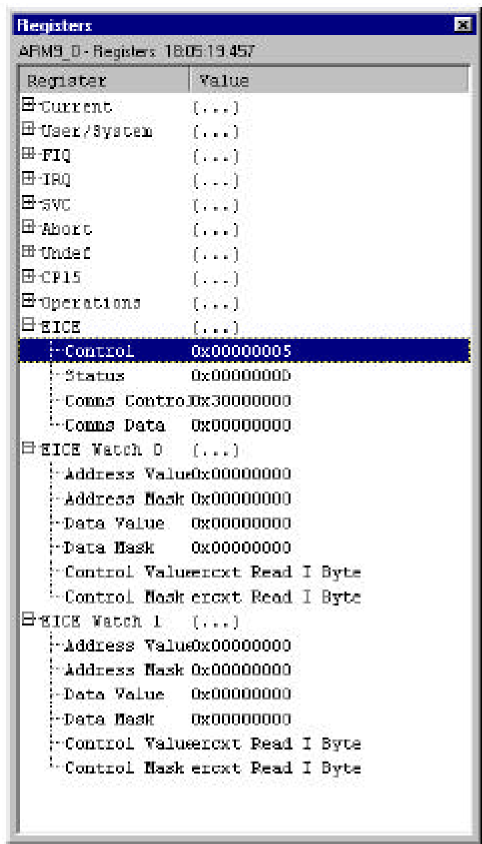


图 47　EmbeddedICE **逻辑寄存器窗口**

AXD **命令行接口命令** reg **按组命名方式显示寄存器。组名称随** AXD **的版本不同而变化：**

ADS 1.0.1　　　　　　　　**名称为** Coproc 0

ADS 1.1 或稍后版本　　　　**名称为** EICE

比如，在 ADS 1.2 **中，命令** `registers EICE` **将显示** EmbeddedICE **逻辑寄存器的内容，如下面的例子：**

```
Debug >reg EICE
Registers Bank: EICE
Index    Name     Value
#1       c0       0x05
#2       c1       0x0D
#3       c2       0x1F
#4       c4       0x00
#5       c5       0x703008AD
#6       c8       0x00000000
#7       c9       0x00000000
#8       c10      0x00000000
#9       c11      0x00000000
#10      c12      0x0000
#11      c13      0x00
#12      c16      0x00000000
#13      c17      0x00000000
#14      c18      0x00000000
#15      c19      0x00000000
#16      c20      0x0000
#17      c21      0x00
```

例　　　　　　　　在 AXD 中显示协作处理器 0 的寄存器

3.12.2 从 ADW 中读取 EmbeddedICE 逻辑寄存器

要使用 ADW GUI 访问协作处理器 0 的寄存器，选择 View→Registers，如图 48 所示。

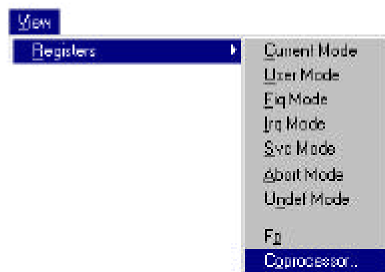


图 48　　查看寄存器菜单

弹出协作处理器对话框。在 Co-processor Number 中键入 0，然后选择 Raw（Unformatted）。如图 49 所示。

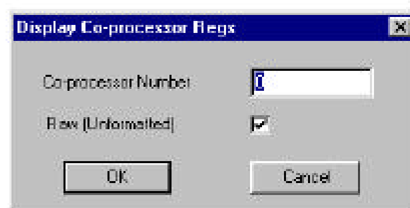


图 49　　显示协作处理器的 Reg 对话框

点击 OK，显示如图 50 所示的协作处理器寄存器窗口。

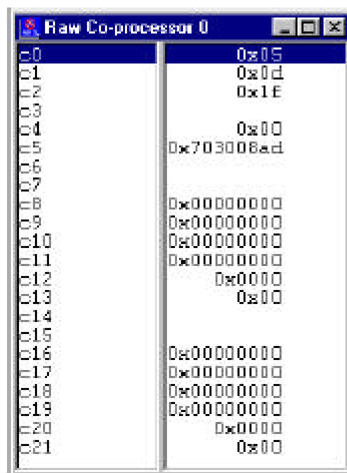


图 50 Raw Co-processor 0 中的 EmbeddedICE 逻辑寄存器视图

ADW 命令 `cregisters 0` 将显示 EmbeddedICE 逻辑寄存器，如下例。

```
Debug: cregisters 0
c0 = 0x05
c1 = 0x09
c4 = 0x00
c5 = 0x00000000
c8 = 0x516ce8da
c9 = 0xbfdf0ea6
c10 = 0xbff6fd7d
c11 = 0xfbaffbff
c12 = 0x0000
c13 = 0xff
c16 = 0x00000008
c17 = 0x00000003
c18 = 0x7dfeeffb
c19 = 0xffffffff
c20 = 0x0100
c21 = 0xf6
```

例　实例协作处理器 0 寄存器的值

3.12.3 使用 EmbeddedICE 逻辑值

位于 EmbeddedICE 逻辑扫描链中的寄存器地址即协作处理器 0 的寄存器号码。因此，你可以很轻松地读取 EmbeddedICE 逻辑寄存器，但向其中写入数据时需要注意。因为 Multi-ICE DLL 也在使用 EmbeddedICE 逻辑寄存器来创建中断点和查看点。当你向一个 EmbeddedICE 逻辑寄存器中写入数据时(比如,使用 ADW 命令 `cwrite 0 20 0x44`),Multi-ICE DLL 将检查中断点寄存器是否在使用中。如果在使用中，Multi-ICE DLL 尝试通过将硬件中断点降级为软件中断点来释放该寄存器。并锁定该中断点寄存器，以避免 Multi-ICE DLL 进一步使用它。

通过显示 icebreaker_lockedpoints 的值，可以找出被锁定的中断点。你还可以将这个变量写到未被锁定的中断点上。在 ARM7 和 ARM9 系列芯片中，中断点号码为 1 和 2，并且 icebreaker_lockedpoints 的 bit 1 和 bit 2 将显示它们的状态。

如果写入协作处理器 0 的一个中断点或查看点被调用，Multi-ICE DLL 将停止其运行，并给

出一个报告信息——Unknown Watchpoint（未知查看点）。这意味着该中断点已经不受 Multi-ICE DLL 控制。

注意：

不要往 EmbeddedICE 逻辑寄存器 0 和 1，控制和状态寄存器中写入数据。Multi-ICE DLL 使用这些寄存器来执行很多它自身的操作，如果被修改，将可能导致数据丢失或 DLL 不能运行。

调试器读写 EmbeddedICE 逻辑寄存器的请求指令并不会使寄存器立即被读取或写入。因为，就效率而言，Smart-ICE 软件只是在程序恢复运行前更新了寄存器内容而已。

3.12.4 ICE 扩展单元的支持

Smart-ICE 具备支持 ARM ICE 扩展单元（IEU）的功能。它是一个可以添加到处理器中的逻辑段，创建好之后能够扩展调试器可用的中断点单元数量。

如果存在此单元，它将被自动调用。IEU 中断点寄存器编号从 2 到 31。其对应的 icebreaker_lockedpoints 段位于 0x4 和 0x80000000 之间。

Part 4. 系统级设计指导

本章介绍如何设计可以使用 Smart-ICE 作调试的、基于 ARM 的设备。具体包含以下几节：

* 总览；
* 系统设计；
* ASIC 设计指导；
* PCB 设计指导；
* JTAG 信号相关性和最大连线长度；
* 调试器接头的兼容性。

4.1 关于系统设计指导

这一节包括以下内容：

如何在系统中连接多 TAP 控制器。比如，将一个 ARM 芯片与数字信号处理器（DSP）相连；

支持分页命令的系统；

使用 Smart-ICE 自适应时钟功能来控制 JTAG 时钟；

复位动信号，提供电路实例；

与 EmbeddedICE 连接头的兼容性。

4.2 系统设计

这一节介绍了如何设计时钟，复位与 Smart-ICE 兼容的电路。包括以下几部分：

* 将 ARM 芯片与其他设计联合使用；
* 使用自适应时钟以达到与 JTAG 端口同步；
* 复位信号。

4.2.1 将 ARM 芯片与其他设计联合使用

你可以使用 Smart-ICE 来调试一个由 ARM 核与其它设备混合的系统。TAP 控制器必须成菊花链式排布（见 " 多设备的 Ics "）。当访问某一特定设备时，Smart-ICE 将把其他的 TAP 控制器置于旁路模式下。在文件 IRlength.arm 中，Smart-ICE 已经设定多个 ARM TAP 控制的寄存器长度。如果没有你需要的，那么必须自己添加。

Smart-ICE还提供一个软件接口层，允许你写访问非ARM设备的驱动。(详情参阅 " *Multi-ICE TAPOp API Reference Guide*. ")

4.2.2 使用自适应时钟（adaptive clocking）以达到与 JTAG 端口同步

基于 ARM 的设备只使用硬件宏单元，比如 ARM7TDMI 和 ARM920T 等，使用的都是标准的五线 JTAG 接口（TCK，TMS，TDI，TDO 和 nTRST)。但是，某些目标系统会要求 JTAG 事件与系统时钟同步。要解决这一问题，就必须在 JTAG 端口上引入一个额外信号。

具体如下：

根据单一的 D 类设计原则设计的 ASIC 信号，比如一个基于 ARM7TDMI-S 芯片中；

一个扫描链位于 ARM 宏单元外部的系统，必须遵循单一的 D 类设计原则。

Smart-ICE 的自适应时钟功能也有这个要求。当启用自适应时钟功能后，Smart-ICE 将生成一个 TCK 信号，并等待 RTCK（返回的 TCK）信号返回。直到收到 RTCK 信号后，Smart-ICE 才处理下一个 TCK 信号。

注意：

如果你使用了自适应时钟功能（adaptive clocking），传输延迟，门电路延迟，同步延迟等将综合导致时钟工作频率降低。除非硬件设计要求使用此功能，否则不要使用它。
在自动配置一个目标硬件时,如果 Smart-ICE 仿真器收到一个 TCK 的返回信号 RTCK，它会认为需要启用自适应时钟功能，并自动打开。如果此时硬件没有要求使用自适应时钟功能，则目标运行速度将会被降低。你应该在 JTAG 设置对话框中禁用自适应时钟功能。

目标板设计的时候，如果需要自适应时钟功能，则在芯片附件将 RTCK 和 TCK 短路，如果不需要自适应时钟功能，可以将 RTCK 直接接地。

你可以使用自适应时钟功能（ adaptive clocking）作为与带有较慢或频幅较大的目标板的接口。比如说一个由电池供电的设备，其时钟速度随操作流程而变化。在这个系统中，TCK 可能比系统时钟快几百倍，这时调试器会失去与目标系统同步。此时，自适应时钟功能就保证JTAG端口速度自动调节系统速度。

图 51 显示了基本应用的电路，而图 52 则显示了该电路的部分时序。通过对系统时钟反向触发器的计时可以减少延迟，因为第二个触发器只解决亚稳定度问题。甚至一个单边触发同步器也从不错过 TCK 信号，因为 RTCK 是控制 TCK 的反馈电路的一部分。

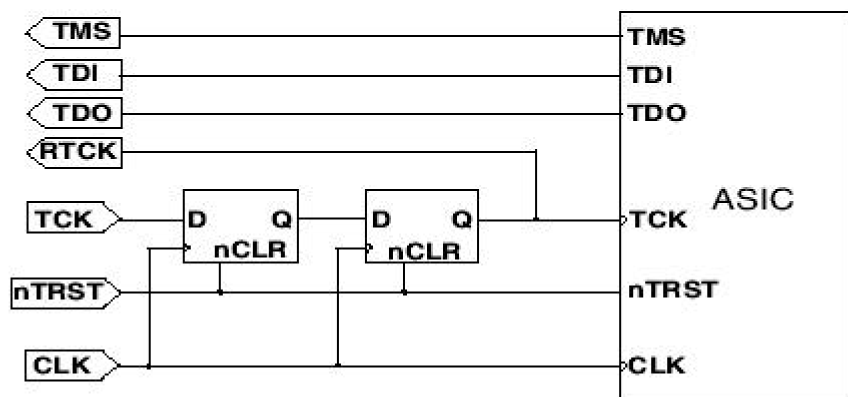


图 51　　基本的 JTAG 同步器

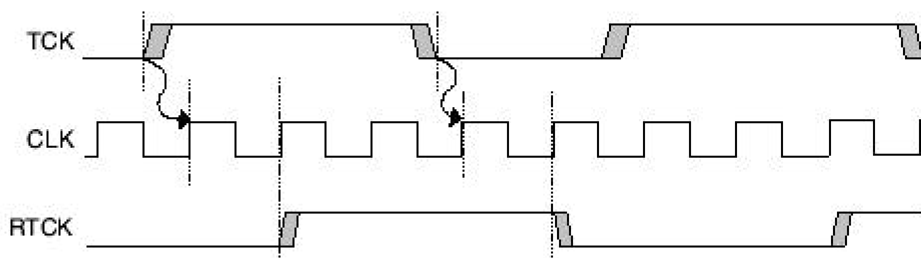


图 52　　基本 JTAG 同步器的时钟调节图

对于一个 ASIC 设计流程和原则而言，设定一个限制条件后，可以通过一个单边时钟对设计中的所有触发器进行计时。要使其 JTAG 端口完全与系统异步，就需要将 JTAG 的 TCK 转换到这个单边时钟里，并确认 JTAG 端口不超过该同步延迟。图 53 显示此电路的一个实例，而图 54 则显示了其部分时钟调节图，清楚地标注出 TCKFallingEn 和 TCKRisingEn 在一个 CLK 周期中是任何各自运行的流程。它还显示了如何启用门电路信号 RTCK 和 TDO，在 TCK 处其状态如何变化。

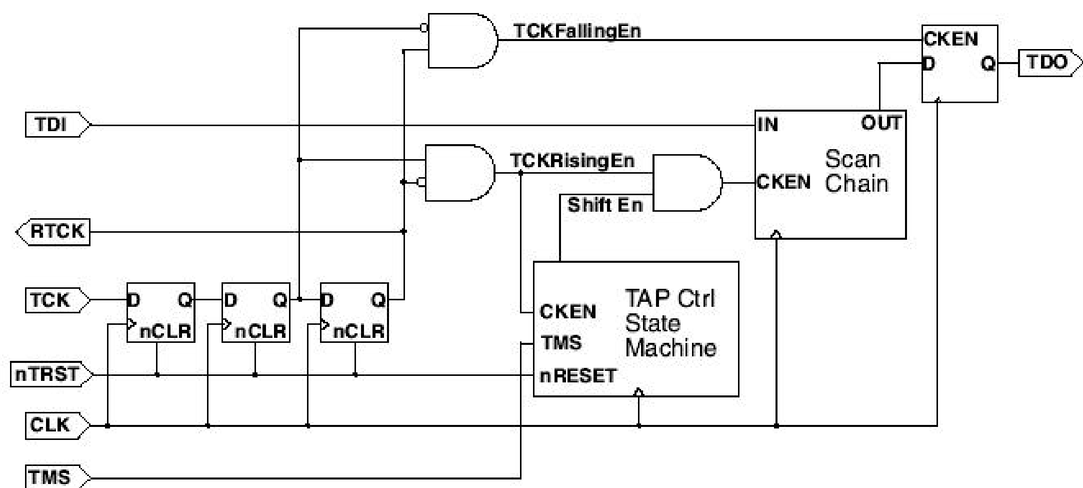


图 53　　遵循单一的 D 类 ASIC 设计原则的 JTAG 端口同步器

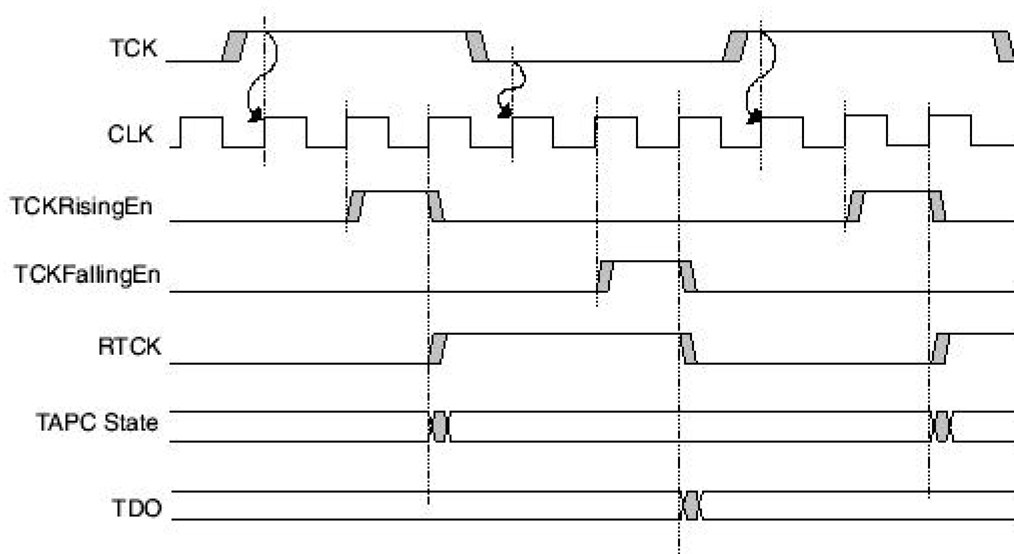


图 54　　D 类 JTAG 同步器的时钟调节图

4.2.3 复位信号

这一节介绍了 ARM 设备上可用的复位信号，以及 Smart-ICE 如何将它们布线的。分为以下几部分：

* ARM 复位信号；
* Smart-ICE 复位信号；
* 复位电路实例。

ARM 复位信号

所有的 ARM 芯片都有一个主复位信号，可能名为 **nRESET**，**BnRES** 或 **HRESET**。它由下列一个或多个条件触发：

通电；
按键复位；
从调试器上远程控制复位（使用 Smart-ICE）；
看门狗电路（如果与应用程序匹配）。

任何包含 JTAG 接口单元的 ARM 芯片都有一个第二复位变量，名为 **nTRST**（TAP 复位）。它将复位 EmbeddedICE 逻辑，TAP 控制器和边缘扫描单元。它由下列一个或多个条件触发：

通电；

远程 JTAG 复位（从 Smart-ICE 上）。

建议这两个信号都可以独立地在 JTAG 连接头上使用。如果 **nRESET** 和 **nTRST** 信号连接起来，则在复位系统的同时将复位 TAP 控制器。这意味着：

不可能通过复位来调试一个系统，因为任何先前设定的中断点都已经丢失了；

你可以在启动阶段开始调试进程，因为 Smart-ICE 在 TAP 控制器状态改变后还没有正式启动。

Smart-ICE 复位信号

Smart-ICE 仿真器有两个复位信号连接到调试的目标板上：

nTRST 驱动 ARM 芯片上的 JTAG **nTRST** 信号。它是一个集电极开路的输出信号，只要 Smart-ICE 软件对目标系统的调试接口进行重新初始化操作时，它就会启动；

nSRST 是一个单向信号，可以驱动并探测目标板上的系统复位信号。其集电极开路的输出模式由调试器设置为 LOW，从而对目标系统进行重新初始化。

这两个复位信号电路中必须包含一个上拉电阻。

RESET 电路实例

图 55，56 显示了实现复位信号的电路以及其操作的形成。图 56 中使用的 MAX823 是一个典型的电压监控器。它有一个由 Smart-ICE 仿真器驱动的 **nRESET** 输出限制电平。

当 Multi-ICE server 探测到一个复位信号，它将记录下此事件，以方便用户了解系统状态。每个当前活动连接均有相应记录，因此一个用户不能阻止其他用户查看复位信息。如果目标系统使用的是监控器复位电路时，将非常有用——因为除此之外，没有其他系统复位的信息了。

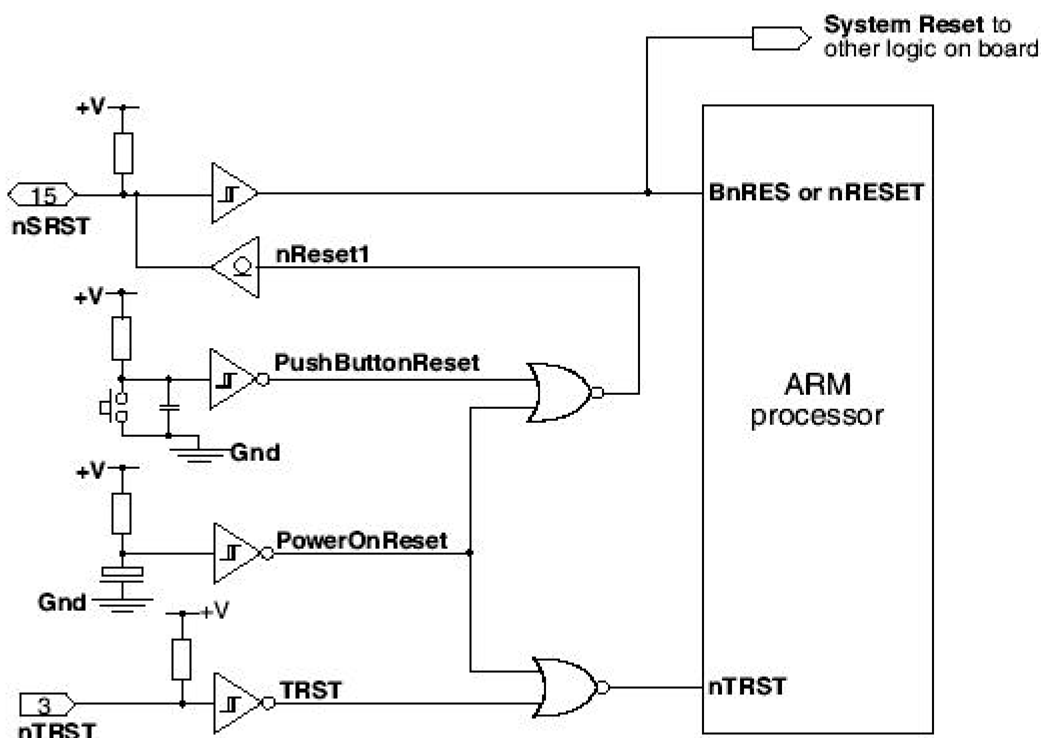


图 55　　复位逻辑电路实例

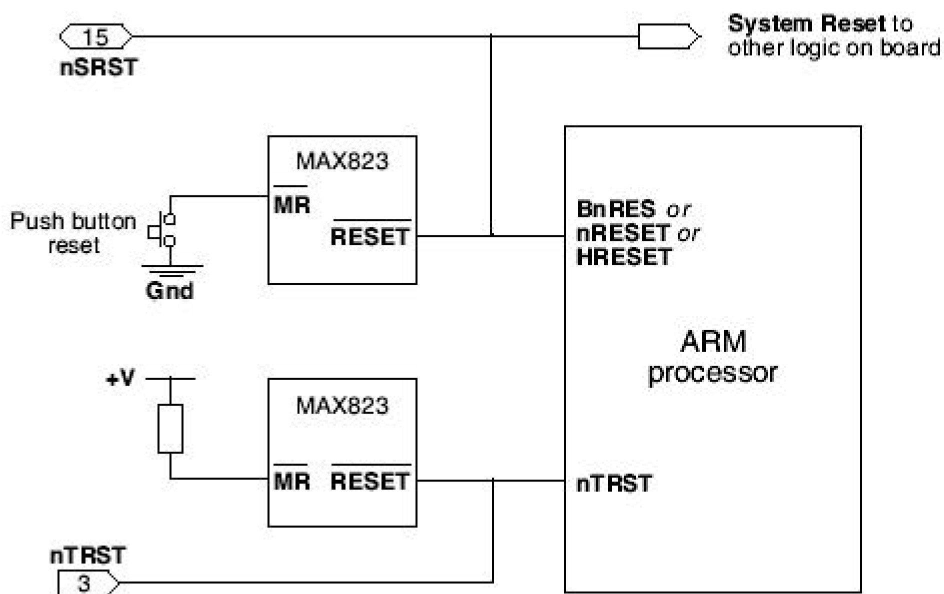


图 56　使用电压监控芯片的复位电路实例

4.3　ASIC 指导

这一节包括：

包含多设备的 ICs**；**

Multi-ICE server **使用的限制条件；**

边缘扫描的测试扇区。

4.3.1 包含多设备的 ICs

JTAG 标准最初认为是 PCB 上一系列菊花链式的多个设备的集成体。现在这一概念则扩展为在单个封装中的多个芯片组合。如果在你的 ASIC 上有不止一个 JTAG TAP 控制器，则它们必须按顺序排列，以方便 Smart-ICE 同时访问所有的芯片。这种链式结构既可以在 ASIC 内，也可以放在其外部。

有以下几种可能的多 TAP 控制器的配置方案：

* 位于 ASIC 中的串连式 TAP 控制器；
* 每个 JTAG 连接都独立地支出引脚；
* 数据信号的合成。

位于 ASIC 中的串连式 TAP 控制器

这是 JTAG 板内连接的自然延伸，也是 Smart-ICE 的推荐方案。其封装包的引脚数并没有增加，对速度影响也非常小——因为没有分配地址的 TAP 控制器可以置于旁路模式下，如图 57 所示。

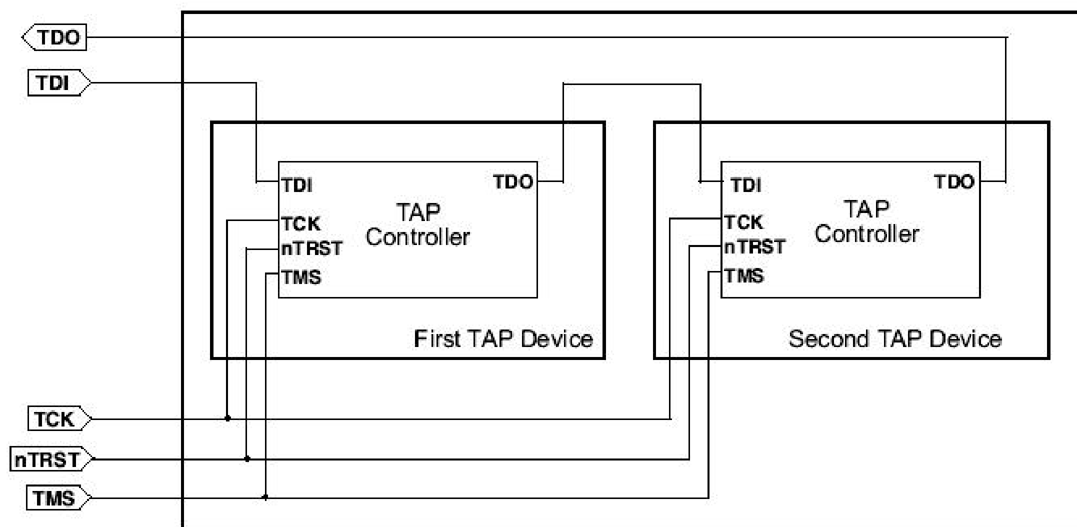


图 57　　　位于 ASIC 中的串连式 TAP 控制器

每个 JTAG 连接都独立地支出引脚

此方案将给予 PCB 设计最大的灵活性，但主要指设备封装包的引脚数。如果此方案用作简单设备的测试，则在使用 Smart-ICE 时，JTAG 端口必须在 PCB 上以串连形式排列。这些独立的 JTAG 端口对应 PCB 上的独立引脚，但要求每个引脚都有对应的 Smart-ICE 的接口单元（实际上是不必要的）。

数据信号的合成

在一系列不同的芯片间，Smart-ICE 还不支持 TCK，TMS，TDI，TDO 和 RTCK 信号的合成。

4.3.2 Multi-ICE server 使用的限制条件

Multi-ICE server 有一些设计上的限制条件，可能会影响到调试系统的能力。

以下限制条件针对所有的系统：
每个 TAP 控制器的 IR（指令寄存器）必须在 2 到 63 位长之间（包含 2 和 63 在内）；
IR 扫描链的总长度为 64 位；
TAP 控制器的最大数量为 64 位。

以下限制条件只针对 TAP 控制器包含 SCSR（扫描链选择寄存器）的情况：
每个 TAP 控制器的 SCSR 长度不能超过下列任何数据——
32 位；
65 位——TAP 控制器的数量；
最大扫描链数为 63。

4.3.3 边缘扫描的测试扇区

如果你使用 JTAG 的边缘扫描测试功能来检测产品的测试扇区的话，就必须有独立访问每个 TAP 控制器的路径。这样可以避免在设备中合并多于一个段的测试扇区的麻烦。具体解决方案是接收该合并结构，然后在设备的封闭集合体中设置一个引脚，使设备中各元件进入测试模式。以此可以中断 TDO 和 TDI 信号的内部菊花链式连接，并通过用作其他用途的引脚合成独立的 JTAG 端口输出信号。

4.4 PCB 指导

这一节介绍了目标 PCB 上的物理和电学连接的指导信息：
*PCB 连接；
*目标接口的逻辑级。

4.4.1 PCB 连接

你最好将 JTAG 引脚放置在尽量靠近目标设备的地方，从而将因为长 PCB 线路引起的信号衰减降到最小。
图 58 显示了可能的原理图。

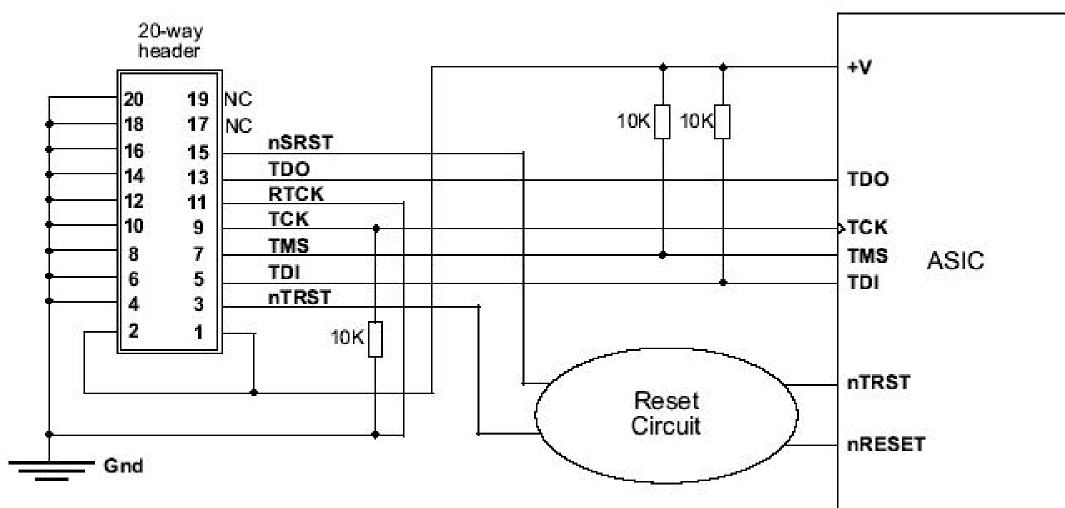


图 58　　典型的 PCB 连接

4.4.2 目标接口的逻辑级

Smart-ICE 可以与多个逻辑等级的目标系统相连。它可以根据目标系统的参考电压来自动调节输出驱动和输入阈值。

VTref（JTAG 连接头的引脚 1）将参考电压传给 Smart-ICE 仿真器。此电压，大概在 3.2V 左右，用作 **TCK**，**TDI** 和 **TMS** 上的逻辑 1 的输出高电平（**Voh**）。0V 用作逻辑 0 的输出低电平。**TDO**，**RTCK** 和 **nSRST** 的输入逻辑阈电压是 **Voh** 的 50%，大概为 1.55V。**Voh** 和 **Vi**（**th**）与 **VTref** 的关系如下图所示。

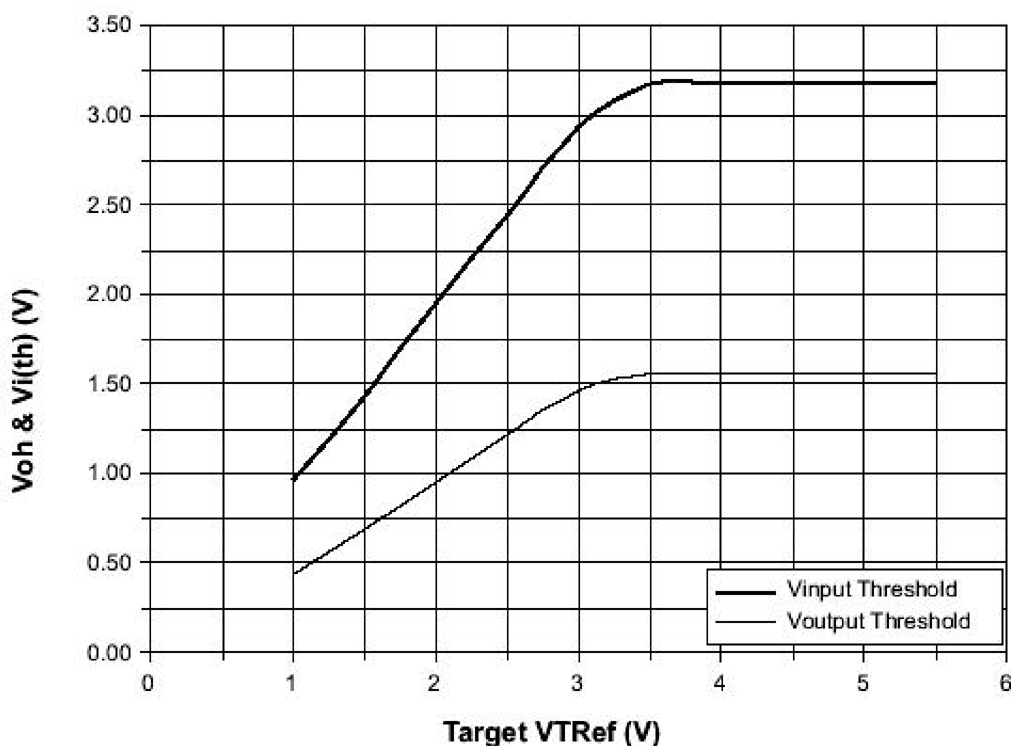


图 59　　目标接口电压等级

工作中的自适应接口电压等级接近 **VTref** 时已经小于 1V。但是，如果 **VTref** 变得低于 0.85V 时候，Smart-ICE 认为这是目标不存在的现象，并且软件将报告一个错误条件的信息。

Smart-ICE 的输出 **nTRST** 将有效地驱动 open collector。它首先被拉到 0V，然后取决于目标系统重的上拉电阻来结束其复位状态。因为这个信号与其他的 **nTRST** 信号（比如说目标系统的断通电复位）共用 OR 线。

Smart-ICE 的输出 **nSRST** 也可以类似地驱动 open collector，并且必须由目标系统的电阻来提升电压。这个信号同样是 Smart-ICE 仿真器的输入信号，而且有一个大阻值的内部拉升电阻（51k? 到 **Voh**）。这将有效避免 **nSRST** 没有同目标系统相连时电平为低的情况。

Smart-ICE 仿真器的输入和输出特性必须符合 TTL 的逻辑兼容标准，或目标系统的 CMOS 逻辑。对于访问其他兼容的逻辑系统的信息，其所有信号的输出阻值大概为 100? 。

4.5　JTAG 信号集成度和最大连线长度

对基于 JTAG 的调试而言，你必须保证 Smart-ICE 与目标板之间的可靠连接。因为没有办法探测或修正错误。就此而言，保证良好的信号质量就显得非常重要了。

限制连线最大长度的因素之一就是信号传送延迟。通常 Smart-ICE 仿真器采集的从目标板上返回的数据会使用输出数据的相同时钟，**TCK**。如果传送延迟过长，Smart-ICE 仿真器将在错误的时间采集数据。解决办法之一是使用自适应时钟。在这种模式下，目标板将返回一个

时钟信号，RTCK，并且 Smart-ICE 不会在 TDO 上进行数据采样，或者将后续数据传送给 TDI，直到由这个信号计时。

在一个 ASIC 或 ASSP（比如，在 ARM 的基于微控制器的芯片中）中，TDO 和 RTCK 信号通常不会使用比其他信号更强的驱动。这些驱动能力随设备而变化。该规格的一个实例是汇点为 4mA。很多设计方案都直接把设备上的这些引脚与 Smart-ICE 接头对应引脚相连。通过一条短连接线，比如由 Smart-ICE 提供的那一条，这类驱动已经足够。但是，如果使用了较长的连接线，则驱动就比较困难了——因为其容载增大了。使用较长连接线，即把它作为一条信号传输线，并要求有匹配的阻抗，否则就会发生信号反射。

Smart-ICE 有很多强力驱动,它们通过一系列 100? 的电阻与 JTAG 连接线达到阻抗匹配。其输出电路可以生成超过 40mA 的电流。这要明显优于目标端口使用的电路。

常见的目标板（驱动能力较差，没有阻抗匹配电阻）上，你只能通过较短的连接线（大约 20cm）才能保证操作的稳定性。如果要使用较长连接线，则必须改进目标板的电路。

推荐的解决方案是：添加一个额外的缓冲器，保证良好的电流驱动，以及一个 100? 的电阻，改善目标板上的 TDO（与 RTCK）信号。通过这一技术，你可以使用长连接线进行调试，大约到几米。取决于连接线长度和缓冲器的传输延迟，有可能使用自适应时钟功能。

如果你在设计中不打算使用自适应时钟功能，则可以让 TCK 信号反馈至同一缓冲器以及 TDO 所使用的阻抗匹配电路，从而在目标板上生成 RTCK 信号。如果要使用长连接线，另一个解决方法就是通过不同的驱动对 JTAG 信号进行缓冲，比如，RS422 和在远端使用双绞线连接到差分接收器。此时就得使用自适应时钟功能以处理连接线和驱动间的传输延迟。使用此技术可以保证在大约几十米长度上的操作稳定性。

降低 Multi-ICE server 的 JTAG 时钟速度可以避免一些，但非全部，与长连接线有关的问题。如果下载代码和在调试器中读取内存的速度变慢不是一个太突出的问题，则可以尝试降低这个时钟的速度。

4.6 与 EmbeddedICE 接口的目标连接头的兼容性

EmbeddedICE 接口单元通常使用 14 位连接头来连接仿真器和目标系统。如有需要，还可以用 20 位连接头连接 EmbeddedICE 接口单元和目标系统。

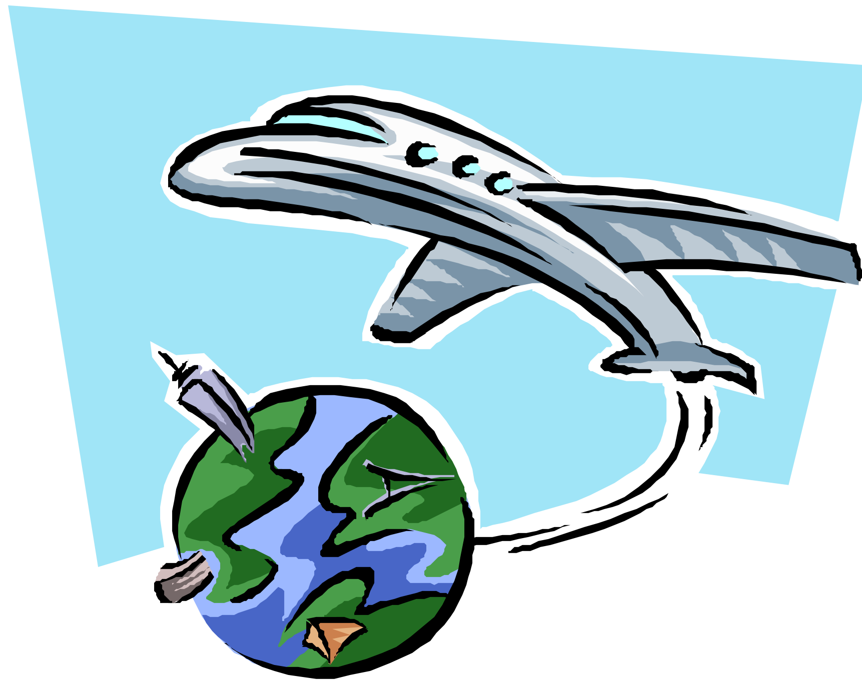
4.6.1 通过转接器连接 Smart-ICE 仿真器和一个 14 位连接头

将转接板上的 14 位插座插入目标板上的对应插头中，同时将 Smart-ICE 的连接线与转接板上的 20 位连接头相连。

转接板上的三脚接头的连接如下：

Pin1	0V
Pin2	Smart-ICE 接头的 pin11，**RTCK**
Pin3	Smart-ICE 接头的 pin9 对应的电阻，**TCK**

连接 PIN1 和 PIN2，则 RTCK 接地，这个时候自适应时钟功能无效，连接 PIN2 和 PIN3。则转接头作为 TCK 到 RTCK 的回流点



上海勤研电子科技有限公司

技术支持热线:021-51097571